Designing recurrent neural architectures for prediction, inference and learning in a long-term memory of sequences using predictive coding

Louis Annabi

Modèles neurocomputationnels pour la prédiction, l'inférence et l'apprentissage au sein d'un mémoire à long-terme de séquences se basant sur le codage prédictif

Designing recurrent neural architectures for prediction, inference and learning in a long-term memory of sequences using predictive coding

Thèse de doctorat pour l'obtention du titre de docteur délivré par CY Cergy Paris Université
Ecole doctorale n°405 Économie, Management, Mathématiques, Physique et Sciences Informatiques (EM2PSI)

Thèse présentée et soutenue à Cergy, le mardi 7 décembre 2021, par
Louis Annabi

Devant le jury composé de :

Verena Hafner	Humboldt-Universität zu Berlin	Rapporteure
Jun Tani	Okinawa Institute of Science and Technology	Rapporteur
Emmanuel Daucé	Institut de Neurosciences de la Timone	Examinateur
Sao Mai Nguyen	ENSTA Paris	Examinatrice
Nicolas Rougier	Université de Bordeaux	Examinateur
Alexandre Pitti	CY Cergy Paris Université	Directeur de thèse
Mathias Quoy	CY Cergy Paris Université	Co-directeur de thèse

Remerciements

Le travail présenté ici n'aurait pas pu être possible sans l'aide de nombreuses personnes qui ont permis à cette thèse de se réaliser, et m'ont soutenu pendant ces trois années.

Pour commencer, cette thèse a été rendue possible grâce à mes encadrants Alexandre et Mathias, que je remercie pour leur aide, leur écoute et leurs conseils au long de ces trois années. Je remercie aussi Nicolas Rougier pour ses conseils au début de cette thèse et lors de la soutenance de mi-parcours.

Je voudrais également remercier mes collègues du laboratoire ETIS: Mehdi, Mingda, Clara, Théo, Zido, Paul, Louis, Sylvain, Mathieu, Lise, Sébastien, Pauline, Aliaa, Julien, Uj, Victor, Mourad et tous les autres, pour les pauses café, les parties de basket, et pour avoir rendu la vie au laboratoire aussi agréable. Merci à Eva, Arnaud, Ghilès et Kévin pour leurs conseils et les longues conversations dans le RER A.

Dans la continuité, je remercie également tous mes amis, de prépa, d'école, et autres, pour les verres, les week-ends, ou les soirées en ligne pendant les confinements, qui m'ont aidé à décrocher un peu du doctorat.

Je remercie Cécilia, et à nouveau Alexandre, Mathias, Sylvain, Eva, Clara, Théo, Mehdi et Mingda pour leur aide sur l'écriture de ce manuscrit, par leurs conseils ou en donnant de leur temps pour relire et m'aider à clarifier certains points.

Sur une note plus personnelle, je voudrais remercier Chloé, Alex, et mes parents Christine et François pour leur soutien pendant ces trois années. Je suis très reconnaissant envers Diem Tu pour avoir partagé avec moi les moments de joie et parfois de déception qui ont accompagnés cette thèse. Enfin, j'aimerais adresser un remerciement spécial à mon grand-père Jean-Paul, qui est la raison principale derrière mon intérêt pour la recherche scientifique.

Contents

List of Figures

List of Tables

List of Abbreviations

AIF Active Inference. 102

ANNs Artificial Neural Networks. 8

BP Backpropagation. 11

BPTT Backpropagation Through Time. 12

CI Chaotic Itinerancy. 81

CNNs Convolutional Neural Networks. 29

ELBO Evidence Lower Bound. 16

ERS Error Regression Scheme. 36

FEP Free Energy Principle. 14

PC Predictive Coding. 8

RC Reservoir Computing. 13

RL Reinforcement Learning. 102

RNN Recurrent Neural Network. 9

VAEs Variational Auto-Encoders. 13

VFE Variational Free-Energy. 14, 16

List of Symbols and Notations

Symbol	Name	Description
	General notations	
s	Scalars	
$\boldsymbol{v}$	Vectors	
$\boldsymbol{M}$	Matrices	
$\boldsymbol{M}^\intercal$	Transpose of $\boldsymbol{M}$	
$\cdot$	Dot product	
$\odot$	Element-wise product	
$\nabla_{\boldsymbol{v}} s(\boldsymbol{v})$	Gradient	Vector composed of the partial derivatives $\frac{\partial s(\boldsymbol{v})}{\partial v_i}$
$\mathcal{N}(\cdot; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Normal distribution	Multivariate normal distribution of mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$.
$\mathbb{E}_p[\cdot]$	Expectation	Expectation with regard to the distribution p.
$D_{KL}(p\|q)$	KL divergence	Kullback-Leibler divergence between p and q.
	Predictive Coding	
$\boldsymbol{X}$	Observation	The observable random variable that the agent tries to predict.
$p(\boldsymbol{x})$	Prior density on $\boldsymbol{X}$	The agent's probabilistic prior belief on the observation.
$\boldsymbol{H}$	Latent state	The hidden random variable representing the state of the environment.
$p(\boldsymbol{h})$	Prior density on $\boldsymbol{H}$	The agent's probabilistic prior belief on the latent state.
$p(\boldsymbol{x}\|\boldsymbol{h})$	Likelihood density	The agent's probabilistic model of how the latent state maps to its observation.
$q(\boldsymbol{h})$	Recognition density on $\boldsymbol{H}$	The agent's probabilistic approximate posterior belief on the latent state.
$\boldsymbol{m}_h$	Recognition density mean	Mean of the recognition density $q(\boldsymbol{h})$, activation of the hidden layer according to the predictive coding framework.
$-\log p(\boldsymbol{x})$	Surprise	Surprise or negative log evidence of the observation $\boldsymbol{x}$.
$F(\boldsymbol{x}, \boldsymbol{h})$	Variational free-energy	Upper bound on surprise, this is the quantity minimized according to the free-energy principle.
	Active Inference	
$\boldsymbol{o}$	Sensory observation	Sensory observation, of dimension d_o.
$\boldsymbol{m}$	Motor command	Motor command, performed by the agent, of dimension d_m.

Abstract

In order to learn and recognize sequences, robotic agents should be equipped with a long-term memory of temporal patterns. Recurrent neural networks are naturally fit for the generation of temporal patterns, and thus can be used to model such a sequence memory using a connectionist approach. Writing in the sequence memory would be tightly related to the question of synaptic weights learning, and memory retrieval could be cast into a problem of inference of the latent causes in the neural generative model. There are many underlying questions to the modeling of this sequence memory. Could it be trained incrementally with minimal forgetting of previously learned sequences? How many temporal patterns could be written in the memory? How to learn motor sequence memories without direct supervision in the motor space? How to retrieve previously learned temporal patterns?

We propose to approach these questions by devising sequence memory networks within the frameworks of predictive coding (Rao and Ballard, 1999) and free-energy principle (Friston and Kilner, 2006), equipped with learning and inference mechanisms for the writing and retrieval of temporal patterns. Throughout this thesis, we apply our models to the learning of handwriting trajectories for a simulated robotic agent. The main contributions brought by this thesis are the following: first, we design recurrent neural networks based on the free-energy formulation of predictive coding. Second, we propose memory retrieval algorithms for sequence memories. Finally, we combine these models with active inference to build sequence memory models able to learn motor trajectories in the absence of direct motor supervision. Part of this work has been compiled in conference and journal publications (Annabi et al., 2020; Annabi et al., 2021b; Annabi et al., 2021a).

Résumé français

Pour être capable d'apprendre et reconnaître des séquences, un agent robotique doit être équipé d'une mémoire à long terme pouvant contenir ces motifs séquentiels. Les réseaux de neurones récurrents sont naturellement adaptés à la génération de signaux temporels, et peuvent donc être utilisés afin de modéliser une telle mémoire de séquences avec une approche connexionniste. Le processus d'écriture dans la mémoire de séquences serait alors lié à la question de l'apprentissage des poids synaptiques, et le processus de remémoration pourrait être formulé comme un problème d'inférence de causes latentes dans le modèle génératif neuronal. Il y a plusieurs questions sous-jacentes à la modélisation de cette mémoire de séquences: Peut-elle être entraînée de manière continue avec un oubli minimal des séquences précédemment apprises ? Combien de motifs séquentiels peuvent être écrits en mémoire avant saturation ? Comment apprendre une mémoire de séquences motrices en l'absence de supervision dans l'espace moteur ? Comment retrouver des motifs temporels précédemment appris ?

Dans cette thèse, nous proposons d'approcher ces questions en concevant des modèles neuronaux de mémoires de séquences s'inspirant de la théorie du codage prédictif (Rao and Ballard, 1999) et du principe de l'énergie libre (Friston and Kilner, 2006), équipés de mécanismes d'apprentissage et d'inférence pour l'écriture et la récupération de motifs séquentiels. Tout au long de cette thèse, nous appliquons nos modèles à l'apprentissage de trajectoires d'écriture manuscrite pour un agent robotique. Les principales contributions apportées par cette thèse sont les suivantes: premièrement, nous concevons des modèles de réseaux récurrents à partir de la formulation du codage prédictif se basant sur la théorie de l'énergie libre. Deuxièmement, nous proposons des algorithmes simulant le processus de remémoration dans des mémoires de séquence. Enfin, nous combinons ces modèles avec l'inférence active pour construire des modèles de mémoire de séquences capables d'apprendre des trajectoires motrices sans supervision motrice directe. Une partie du travail présenté a été compilée dans des publications en conférences (Annabi et al., 2020; Annabi et al., 2021a) ainsi que dans un article de journal (Annabi et al., 2021b).

Chapter 1

Introduction

1.1 Background

In order to survive, living systems have to maintain their biological constants within certain ranges. This property, called homeostasis, can be formulated as a natural resistance to change and is ensured by different biological mechanisms. Extending this idea to cognitive systems, the free-energy principle (Friston and Kilner, 2006) posits that living agents try to maintain their sensory states within certain bounds, determined by their model of the world. For instance, these models could dictate some preferred interoceptive sensory states such as not feeling hungry or some preferred exteroceptive sensory states such as tasting something good. The extent to which sensory states differ from their predictions can be measured by an information-theoretic quantity called *surprise*. In the previous examples, feeling hungry or eating something that tastes bad would result in surprising sensory states, that the living organism should try to avoid. According to the free-energy principle, several cognitive functions can be framed as homeostatic mechanisms minimizing this surprise. As such, perception is explained as an inference process minimizing surprise by improving one's internal representations of the world. Similarly, learning rules in the internal models can be seen as implementing a process of minimization of surprise. Going further, motor control and decision-making can also be included in this theory: in order to avoid surprise, organisms actively seek out states of their environments that provide the predicted sensory states.

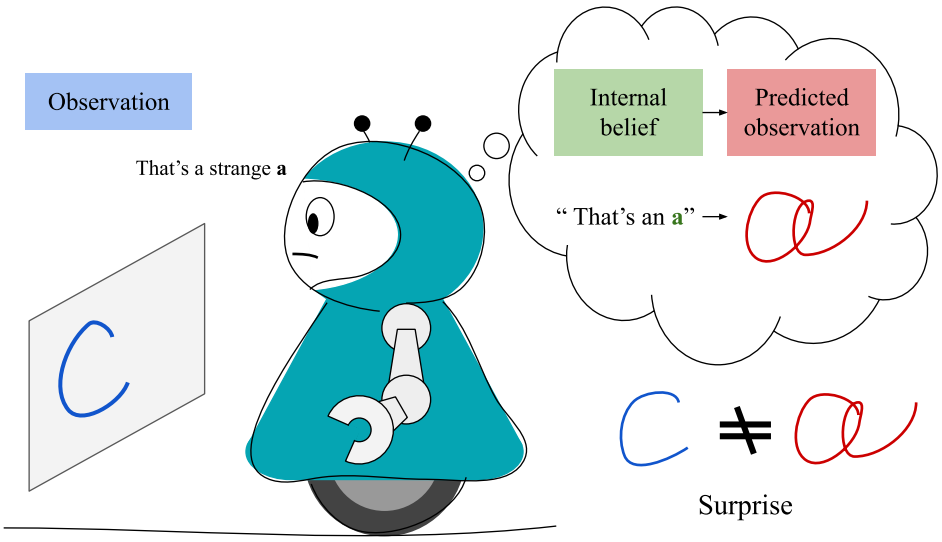


Figure 1.1: An illustration of the concept of surprise.

In particular, the predictive coding theory (Rao and Ballard, 1999) describes one possible implementation of this principle for perceptual inference and learning. In this implementation,

internal neural models actively try to predict sensory states, and neural populations encoding surprise (more precisely prediction error) are used to update the representations of the world encoded in the internal models (inference), as well as the parameters of these models (learning).

Figure 1.1 illustrates the idea of *surprise*. A cognitive agent is faced with a visual observation of the letter c, however, its initial internal belief is that it should see the letter a. The discrepancy between its visual idea of an a (its prediction), and what it actually sees (its observation), gives rise to a feeling of surprise, that the agent tries to minimize. The most natural way to minimize surprise, in this case, would be to update its internal belief to acknowledge that it sees a c instead of an a. This is the process of perceptual inference. Another less intuitive but still valid way of minimizing surprise would be to update its internal model of what an a looks like. Updating this model so that it predicts the image of a c with the internal belief of seeing the letter a would minimize surprise, since the visual observation predicted by the agent when thinking of an a would no longer differ from the current observation. Additionally, a more indirect way to minimize surprise would be to actively fulfill the expectation of seeing an a, for instance by writing one on the white board. All these processes of surprise minimization are at the center of the work presented in this document.

Specifically, in this thesis, we question whether long-term memories of temporal patterns could be modeled using this theory. A long-term memory is a cognitive system storing learned (or written) patterns indefinitely. These patterns can be retrieved by querying the long-term memory with a certain input, that we can call key, or with an approximate version of a learned pattern. In particular, we are interested in long-term memories of *temporal* patterns, that we call sequence memories. For example, such systems can be used to store and retrieve temporal signals such as pieces of music, sentences, or movements. The main goal of this thesis is to propose sequence memory models based on the free-energy formulation of predictive coding, and associated mechanisms for prediction, inference, and learning.

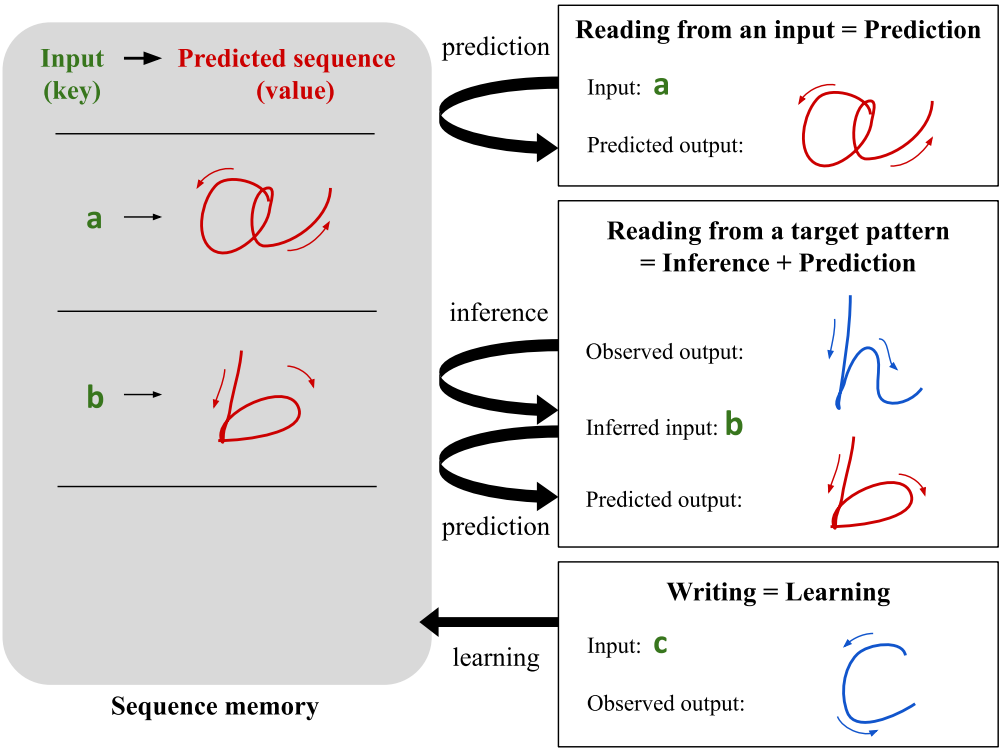


Figure 1.2: Example of a sequence memory of handwriting trajectories.

An example of sequence memory is represented in figure 1.2. This sequence memory contains sequences of 2D points corresponding to learned handwriting trajectories. To design a proper sequence memory model, we must build mechanisms for writing and reading in the memory. Writing in the memory corresponds to the action of inserting a new sequential pattern in the memory, while reading in the memory can be implemented in two ways:

- First, the memory can be queried using an input (or key) that the sequence memory can

directly use to generate the desired sequence pattern.

- Second, the memory can be queried using a possibly corrupted version of the desired sequence pattern. For instance, in the example presented in figure 1.2, the query corresponds to a trajectory depicting an h. This trajectory is similar to the trajectory corresponding to a b that is written in the sequence memory. The sequence memory model should be equipped with inference mechanisms allowing to retrieve the most likely input, here the input "b". This input (or key) can then be used to generate the corresponding sequence pattern.

Predictive coding seems like a promising candidate framework for modeling such cognitive systems, as the question of writing in the memory can be cast as a learning problem, and the question of reading from the memory can be cast as an inference problem.

1.2 Problem definition

We start by providing a more detailed and formal definition of sequence memories. First of all, as described previously, this type of memory falls in the class of long-term memories, as opposed to short-term memories. To grasp a proper understanding of long-term memories, we also need to briefly describe short-term memories. The task of a short-term memory is to store over time information provided as input. We can note the difference between simple short-term memories, and working memories, often considered as processing the incoming information on top of its retention role. In the case of neural networks, this information is typically stored in the network activations. This is consistent with neuroscientific studies showing groups of neurons firing for some time after the presentation of a stimulus (Fuster, 1973).

In contrast, long-term memories do not encode information about past inputs but encode a function associating an input (or stimulus, or key) with an output (or response, or value). Cognitive psychology considers several types of long-term memories: episodic memory, semantic memory, and procedural memory. Episodic memories encode episodes, that is, past events. Semantic memories encode concepts, facts, or knowledge. Episodic and semantic memories are grouped under the name of declarative or explicit memories. Finally, procedural memories encode motor sequences, or gestures. In neuroscience, long-term memories are thought to be encoded in synapses (Dudai, 2004), which translates to weight parameters in artificial neural networks. The process in which an item is encoded in a long-term memory is called synaptic consolidation. Once encoded in synapses, the item is held indefinitely. Still, we can note two situations when forgetting can occur. First, if the neural synapses are modified through other processes happening in the brain. In artificial neural networks, this can happen when training a model on new patterns. Second, if the input used to stimulate the long-term memory is itself the result of some neural processes that have been modified, the (key, value) pair might still be properly encoded in the long-term memory but would be impossible to retrieve because the key is lost.

More formally, we define a long-term memory as a function $f : \mathcal{C} \to \mathcal{X}$ associating the input $\mathbf{c} \in \mathcal{C}$ with the output $\mathbf{x} \in \mathcal{X}$. Stimulated with different inputs, the memory outputs different responses. Adding on top of this definition the sequential nature of the output, we obtain the following definition of a sequence memory, which we use throughout this thesis:

Definition A sequence memory is a function $f : \mathbf{c} \in \mathcal{C} \to (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T) \in \mathcal{X}^T$ associating a stimulus $\mathbf{c} \in \mathcal{C}$ with a sequential response $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$.

We can identify two key dimensions of a sequence: the dimension T that we call temporal dimension, or simply *length of the sequence*, and the dimension of the vector space $\mathcal{X}$, denoted $|\mathcal{X}|$, that we call feature dimension, or simply *dimension of the sequence*. We use in our experiments sequences of dimension 2 and length 60. In this case, our sequence memory is a function $f : \mathcal{C} \to (\mathbb{R}^2)^{60}$.

Other definitions of long-term memories, such as the Hopfield network (Hopfield, 1982), or the sparse distributed memory model (Kanerva, 1988), consider the second reading task of generating a stored pattern when queried with a corrupted version of this stored pattern. This question of memory retrieval is investigated in chapter 5, until then, we stick to the simpler definition provided above.

There are many approaches to the question of sequence memory learning, dealing with different issues. Here, we introduce and explain the issues faced by these systems.

- **Capacity:** capacity is a measure of the information quantity contained in the sequence memory. This quantity is often more informative when compared with a measure of the model complexity. For instance, we can measure the ratio of sequence patterns number properly learned per model parameter. According to this definition, a sequence memory A with 1000 parameters that can store 10 patterns is inferior in capacity to a sequence memory B with 500 parameters that can store 9 patterns. This comparison is based on the assumption that by multiplying by 2 the number of parameters in the sequence memory B we would be able to store 2 times more patterns. More generally, this assumption states that the capacity measure we defined is independent of the number of model parameters, and only depends on the model itself. This assumption will be experimentally tested and we will use this measure to confront different sequence memory models.

- **Continual learning:** during training, a model can forget a part of the previously learned information, a phenomenon labeled catastrophic forgetting. An ideal model should ally plasticity, to be able to incorporate new information, and stability, to avoid the forgetting of already stored information. In consequence, we will question the possibility of an incremental training of our models, and evaluate them according to their ability to learn new sequence patterns without forgetting previously learned patterns.

- **Inference:** according to our definition, a sequence memory is a function associating a sequence with a given input. The problem of inference is the recovery of the input corresponding to a given sequence. In other words, this problem is that of the inversion of the function f modeled by our sequence memory. If we consider the case where $\mathbf{c}$ is an integer taking values in a finite set, associating a category $\mathbf{c}$ with a given sequence is a well-studied question in machine learning, that of sequence classification. In our case, we will try to evaluate to what extent our sequence memory models can serve as models for sequence classification.

- **Supervision:** in the simpler setting that we explore during most of this thesis, we assume that the sequential patterns to be memorized are directly available for the learning system. However, when we consider the problem of motor trajectories learning, supervision in the motor space might not be directly available to the agent. In order to learn a motor sequence memory, additional mechanisms should be proposed in order to build the motor patterns that should be stored in memory, according to indirect criteria. The criteria driving decision could be the research of extrinsic rewards, as in the reinforcement learning literature, or indirect supervision via desired sensory states, an approach more in line with the free-energy principle.

- **Biological inspiration:** the goal of this thesis is not to build models of brain function, but to propose models performing well on the criteria introduced before. However, a lot of progress in artificial intelligence and in machine learning has been achieved by taking inspiration, to some extent, from theories about brain function. The approach at the center of this work is to build upon one of these theories: predictive coding. As such, we will sometimes inquire about the likelihood of our models being actually implemented in the brain.

1.3 Contributions

The artificial neural networks literature overflows with models capable of generating sequences. While feedforward architectures such as convolutional neural networks or attention-based models can be used to generate temporal data, recurrent neural networks are often the models of choice to deal with sequential tasks. In particular, there exist some predictive coding-based recurrent neural network implementations in the literature, which show that it is possible to model sequence memories by taking inspiration from the predictive coding theory. However, these models do not fully address the different issues we have just listed, and additional work is needed to better assess the viability of the predictive coding theory on all these questions.

Our methodology consists in systematically starting from the free-energy principle mathematical framework to address these questions. The proposed models are formally derived from this

framework and then compared to other models from the literature on the different criteria described above. Another source of inspiration for the work presented here is the INFERNO model formerly proposed by our team (Pitti et al., 2017). The general objective of this thesis is primarily theoretical, and we do not aim at applying our models in any specific domain. As such, we have identified a simple yet complex enough task on which we test our algorithms: we choose to build sequence memories of simple 2D continuous trajectories. The simplicity of this task presents several benefits, as it entails low computational costs, easy visualization of the sequence patterns, and a straightforward extension to motor trajectories (for instance handwriting with a robotic arm).

The contributions of this thesis are:

- An overview of the literature on recurrent neural networks and predictive coding in chapter 2, as well as on the intersection of these two subjects in chapter 3. Additionally, we present the current state of the art of the research on active inference in chapter 6.
- The design of several recurrent neural network models implementing predictive coding. Based on the free-energy formulation of predictive coding, we derive in chapter 3 six recurrent neural network models with associated algorithms for inference and learning.
- A comparative study of recurrent neural networks' generative capacity for continuous sequences, as well as a comparative study of online learning algorithms for sequence generation in a continual learning setting (in chapter 4).
- The design of retrieval methods for temporal patterns written in a sequence memory based on predictive coding. We show in chapter 5 that the bottom-up inference mechanisms can interplay with a top-down Gaussian mixture prior distribution to entail an iterative algorithm for memory retrieval, and draw a connection with the phenomenon known as chaotic itinerancy in dynamical systems.
- The design of sequence memories for motor patterns using the active inference framework to construct the motor trajectories that should be written in the sequence memory. We propose in chapter 6 a general architecture for integrating active inference in the previously derived sequence memory models, as well as two attempts at implementing this architecture for the learning of arm joint angle trajectories for drawing and handwriting.

Some of these contributions have been compiled in conference and journal publications:

- (Annabi et al., 2020) "Autonomous learning and chaining of motor primitives using the Free Energy Principle", Louis Annabi, Alexandre Pitti, and Mathias Quoy, 2020 International Joint Conference on Neural Networks (IJCNN), 2020, Online.
- (Annabi et al., 2021a) "A Predictive Coding Account for Chaotic Itinerancy", Louis Annabi, Alexandre Pitti, and Mathias Quoy, Artificial Neural Networks and Machine Learning – ICANN 2021, 2021, Online.
- (Annabi et al., 2021b) "Bidirectional interaction between visual and motor generative models using Predictive Coding and Active Inference", Louis Annabi, Alexandre Pitti, and Mathias Quoy, Neural Networks, 2021.

The Ph.D. thesis is organized as follows. In chapter 2, we introduce preliminary concepts related to our approach, such as the necessary artificial neural networks background, and an overview of the predictive coding research field. In chapter 3, we derive recurrent neural network models implementing predictive coding. In chapter 4, we study their generative capacity as well as their ability to extend to a continual learning setting using only online learning mechanisms. In chapter 5, we show how the inference processes at the core of the predictive coding theory can provide powerful memory retrieval mechanisms. In chapter 6, we propose to extend the derived models using the active inference framework in order to build repertoires of motor trajectories. Finally, chapter 7 concludes this document by discussing our findings, the shortcomings of our work, and the open research directions for future work on this subject.

Chapter 2

Context

In this chapter, we introduce the necessary background in Artificial Neural Networks (ANNs), as well as an overview of the Predictive Coding (PC) research area.

2.1 Artificial neural networks

2.1.1 Introduction and notations

ANNs form a category of models widely used in machine learning and originally inspired by biological neurons. A neural network can be represented as a directed graph, where the nodes are neurons, and the edges correspond to synaptic connections. In all the works presented in this thesis, a neuron is considered as a processing unit computing a real-valued output, its activation, based on activations of its parent neurons in the graph.

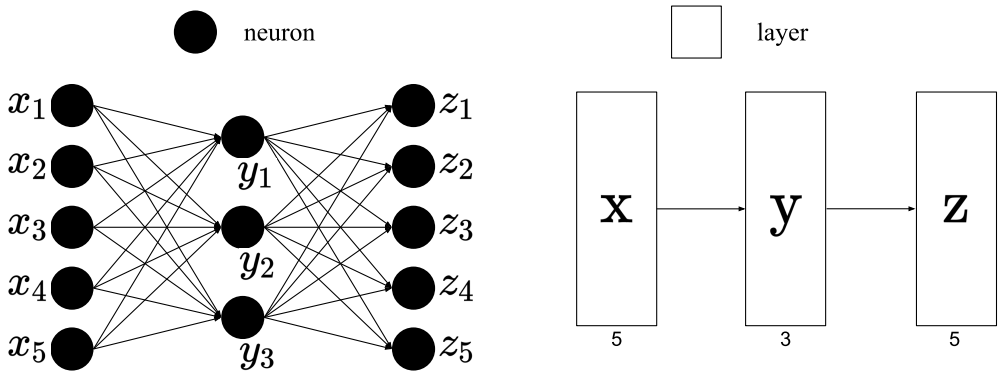


Figure 2.1: Layered representation of a neural network.

The information processed by neural networks being multidimensional, we directly consider layers of a neural network. A layer denotes a group of neurons sharing the same parent neurons. We call activation of the layer the vector of activations of the layer's neurons. In the models presented in this thesis, we represent neural networks by boxes corresponding to the different layers, connected by directed edges. The function performed by each layer shall be explained as part of the model definition.

Figure 2.1 displays two graphical representations of the same network. The representation on the right regroups the neurons into layers. As shown in this figure we sometimes explicitly write the layer dimension in its representation. In some cases (for instance in convolutional networks), the encoded variable has two (matrix) or more (tensor) dimensions, and we shape the layer box representation using a geometry corresponding to this structure (parallelogram, rectangular parallelepiped).

A layer with no parent in the graph is called input layer. For each network, we can also define a set of output variables. In the example figure 2.1, if we choose $\boldsymbol{z}$ to be the output variable, the network can be seen as implementing a function $f : \boldsymbol{x} \rightarrow \boldsymbol{z}$.

There are several approaches for the learning of neural network parameters. This is addressed in section 2.1.3, where we explain and compare different learning algorithms.

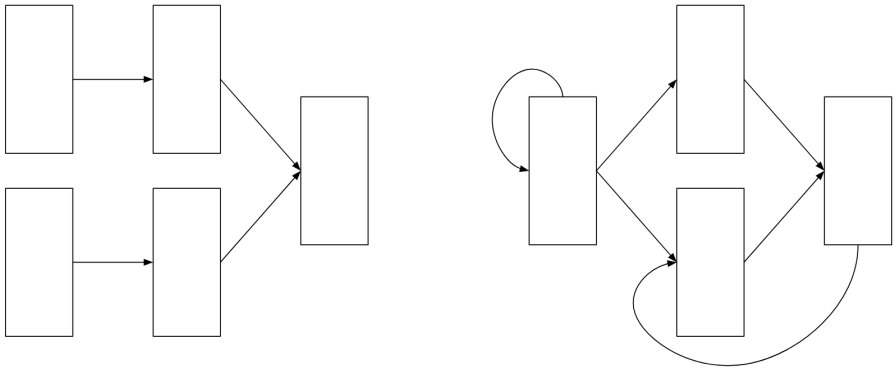


Figure 2.2: Feedforward and recurrent neural networks.

Figure 2.2 displays examples of neural network representations. The graph on the left figure is acyclic, it thus represents a feedforward neural network. In opposition, the graph on the right figure comprises cycles, it thus represents a Recurrent Neural Network (RNN).

2.1.2 Recurrent neural networks

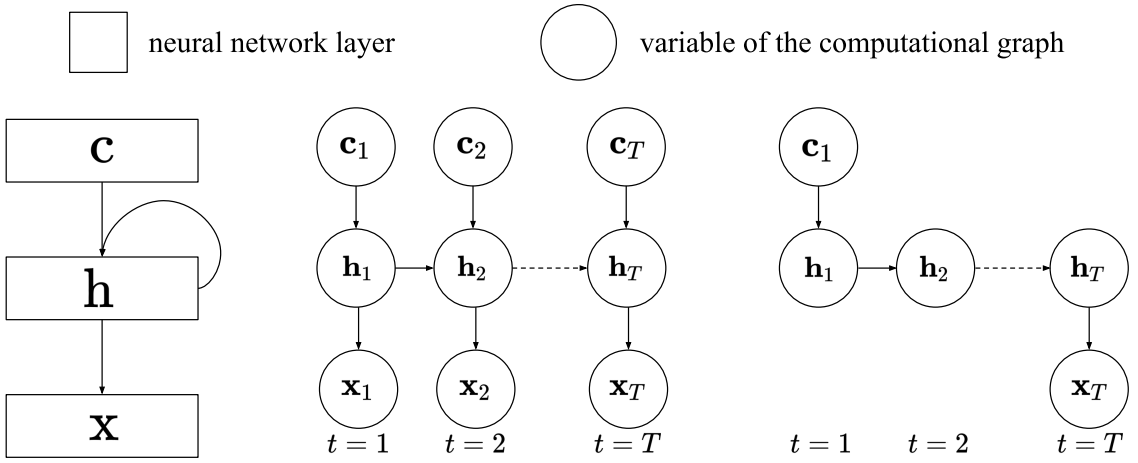


Figure 2.3: RNN and possible corresponding computational graphs.

RNNs refer to the category of neural networks comprising cycles in their directed graph. These models are particularly fit to process sequential data (Elman, 1990). The left graph in figure 2.3 represents a simple RNN, composed of an input layer $\boldsymbol{c}$, a recurrent layer $\boldsymbol{h}$, and an output layer $\boldsymbol{x}$. In the case of feedforward neural networks, the way to compute the output from the model inputs is explicit. The activation of each layer is computed based on the activation of the parent layers. In the case of the RNN represented figure 2.3 though, this is not explicit. The order in which we should use the different synaptic connections to update the activations of the layers is not given. To explain the computations performed by RNNs, we need to provide an associated computational graph. For the network presented in figure 2.3, there are several possible computational graphs, two of them being displayed on the right side of the figure.

In a computational graph, each vertex corresponds to the value of a variable, each variable being computed based on its parents. To easily distinct neural network representations from computational graphs, we use circles to represent vertices in the computational graphs.

The first computational graph supposes the existence of an input sequence of variables $(\boldsymbol{c}_0, \ldots, \boldsymbol{c}_{T-1})$. At every time step t, the activation of the recurrent layer $\boldsymbol{h}_t$ is computed from the input $\boldsymbol{c}_t$ and

the past activation of the recurrent layer $\boldsymbol{h}_{t-1}$. The activation of the output layer $\boldsymbol{x}_t$ is computed from the activation of the recurrent layer $\boldsymbol{h}_t$.

In the second computational graph, the input variable $\boldsymbol{c}_0$ is used to compute the first activation of the recurrent layer $\boldsymbol{h}_0$. Then, during the remaining time steps, the activation of the recurrent layer is computed from its past value. Finally, at the last time step, the activation of the output layer $\boldsymbol{x}_{T-1}$ is computed from the recurrent layer activation $\boldsymbol{h}_{T-1}$.

Based on seemingly the same neural network, we have built in the first case a function associating an output sequence with an input sequence, and in the second case a function associating an item with an item.

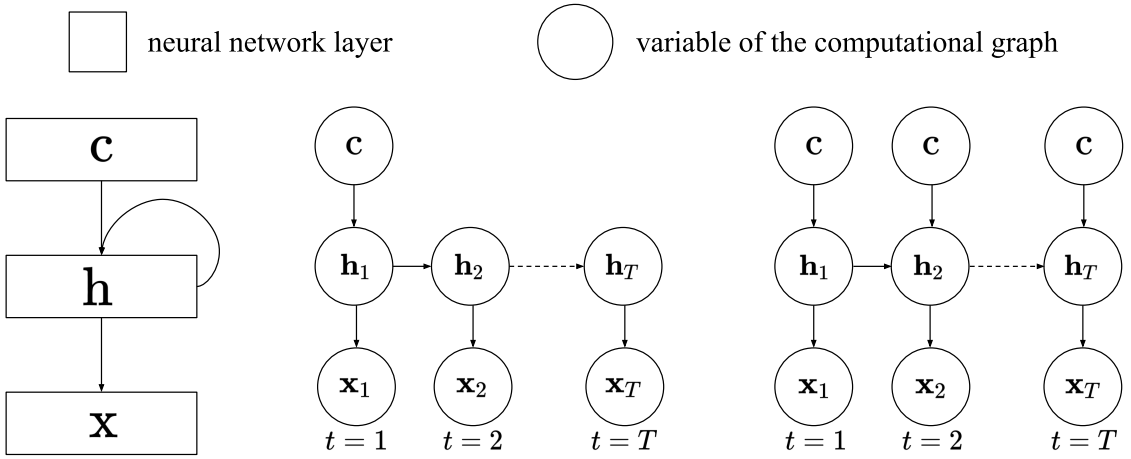


Figure 2.4: Possible computational graph for a sequence memory.

A sequence memory model needs to associate a sequence $(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_T)$ with an integer $k < p$, where p is the number of sequence patterns to learn. We can use the input variable $\boldsymbol{c}$ of dimension p corresponding to the one-hot encoding of k. Figure 2.4 displays two computational graphs that could be used to model a sequence memory using the simple RNN representation we have introduced. In the first computational graph, the input $\boldsymbol{c}$ is only used for the computation of the first activation in the recurrent layer, whereas in the second graph, it is used at each time step.

Here is the set of equations corresponding to the first computational graph:

$$\boldsymbol{h}_0 = \boldsymbol{W}_i \cdot \boldsymbol{c} \tag{2.1}$$

$$\boldsymbol{h}_{t+1} = \boldsymbol{W}_r \cdot \tanh(\boldsymbol{h}_t) + \boldsymbol{b}_r \tag{2.2}$$

$$\boldsymbol{x}_t = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{h}_t) + \boldsymbol{b}_o \tag{2.3}$$

In these equations, $\boldsymbol{W}_i$ is a matrix of dimension (d_h, p) that we call input weights, $\boldsymbol{W}_r$ is a matrix of dimension (d_h, d_h) that we call recurrent weights, and $\boldsymbol{W}_o$ is a matrix of dimension (d_o, d_h) that we call output weights. We have also introduced biases $\boldsymbol{b}_r$ and $\boldsymbol{b}_o$ of respective dimensions (d_h) and (d_o). For simplicity, we omit bias coefficients for the remainder of this thesis.

One interesting feature of this simple model is that it can be obtained from a differential equation guiding the temporal dynamics of the variable $\boldsymbol{h}_t$:

$$\frac{d\boldsymbol{h}_t}{dt} = \frac{1}{\tau}(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{h}_t) - \boldsymbol{h}_t) \tag{2.4}$$

where τ is a time constant. If we use this derivative and estimate $\boldsymbol{h}_{t+1}$ using the first-order Taylor expansion, we get:

$$\begin{aligned} \boldsymbol{h}_{t+1} &= \boldsymbol{h}_t + \frac{1}{\tau}(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{h}_t) - \boldsymbol{h}_t) \\ &= (1 - \frac{1}{\tau})\boldsymbol{h}_t + \frac{1}{\tau}(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{h}_t) \end{aligned} \tag{2.5}$$

In the special case where $\tau = 1$, this equation simplifies to the update rule given in equation 2.2. In fact, we keep this new equation as a generalized version of the simple RNN (sRNN) where

the temporal dynamics are parameterized by the coefficient τ. Basically, high values of τ cause the dynamics of $\boldsymbol{h}$ to be very slow, while the extreme value of $\tau = 1$ provide faster dynamics of $\boldsymbol{h}$. We call sRNN the simple model with $\tau = 1$ and Time RNN (TRNN) the general version parameterized by τ.

This link with differential equations can encourage us to study RNNs from a dynamical systems point of view. In this approach, $\boldsymbol{h}_t$ is seen as the state of the dynamical system, and its temporal evolution is dictated by $\boldsymbol{W}_r$, $\boldsymbol{b}_r$, and the input $\boldsymbol{c}$ (more generally the input can be temporal as well).

Dynamical systems can contain attractors such as point or limit cycle attractors. A point attractor is a stable equilibrium value of the state, trajectories starting from neighboring states converge towards this point. For example, in the presence of friction, a pendulum always converges to a vertical position. A limit cycle attractor is a stable periodic orbit. For example, without friction, the pendulum trajectory is cyclic. However, to be a limit cycle attractor, it would need to attract neighboring orbits as well. These considerations can be of interest when studying the learning of RNN parameters, as we see in section 2.1.3.

A useful tool for the analysis of dynamical systems is the concept of Lyapunov exponent. A dynamical system characterized by a state $\boldsymbol{h}_t$ can for instance exhibit dynamics converging to attractors, or dynamics that are chaotic. We can measure the rate at which two infinitely close states of the system move away from each other. For a multidimensional dynamical system such as an RNN, the maximum Lyapunov exponent is defined as:

$$\lambda(\boldsymbol{h}_0) = \lim_{t\to\infty} \lim_{\boldsymbol{\epsilon}\to 0} \log \frac{|\boldsymbol{h}_t^a - \boldsymbol{h}_t^b|}{\boldsymbol{\epsilon}} \tag{2.6}$$

where $\boldsymbol{h}_t^a$ and $\boldsymbol{h}_t^b$ correspond to the states at time t ensuing an initialization of respectively $\boldsymbol{h}_0$ and $\boldsymbol{h}_0 + \boldsymbol{\epsilon}$. The dynamics of an RNN are determined by its Jacobian matrices $\boldsymbol{J}_t$, defined by:

$$J_t^{ij} = \frac{\partial h_t^j}{\partial h_{t-1}^i} \tag{2.7}$$

Using the Jacobian matrices we can derive a matrix whose eigenvalues are the Lyapunov exponents of the dynamical system. The maximum of these eigenvalues is the maximum Lyapunov exponent, that can provide an idea of the predictability of the system. If this exponent is positive, two neighboring trajectories of the dynamical system tend to move away from each other. Inversely, if it is negative, neighboring trajectories tend to get closer.

These considerations turn out to be useful for the design and analysis of RNN models.

2.1.3 Learning

Deep learning has encountered a large success in the last decade, for example in playing video games (Mnih et al., 2013) and board games (Silver et al., 2016; Silver et al., 2017), in computer vision (Krizhevsky et al., 2012) and in natural language processing (Vaswani et al., 2017; Brown et al., 2020). While all those successes rely on different ANN architectures and learning paradigms, they all use the Backpropagation (BP) algorithm to learn model parameters.

BP (Werbos, 1982; Rumelhart et al., 1986) is an algorithm for training feedforward neural networks. It computes the gradient of the model parameters with respect to a loss function, and thus it can be used to perform gradient descent on the model parameters.

Here is the detailed algorithm applied to a multi-layer perceptron comprising L layers. We denote by $\boldsymbol{W}^{(l)}$, $\boldsymbol{z}^{(l)}$ $\boldsymbol{a}^{(l)}$ $\boldsymbol{f}^{(l)}$ respectively the weights, weighted input (pre-activation potential), activation, and activation function of the l-th layer. The weighted input $\boldsymbol{z}^{(l)}$ is the quantity that, given as input to the activation function $\boldsymbol{f}^{(l)}$, provides the activation $\boldsymbol{a}^{(l)}$. The loss function C associates a real-valued scalar quantity with the activation of the last layer $\boldsymbol{a}^{(L-1)}$.

The gradient with respect to the l-th layer can be computed based on the gradient with respect to the next layer $(l+1)$, according to the following equation:

$$\frac{\partial C}{\partial \boldsymbol{z}^{(l)}} = \left(\left(\boldsymbol{W}^{(l+1)}\right)^\intercal \cdot \frac{\partial C}{\partial \boldsymbol{z}^{(l+1)}} \right) \odot \boldsymbol{f}'^{(l)}(\boldsymbol{z}^{(l)}) \tag{2.8}$$

where $\odot$ denotes the element-wise product. Starting from the gradient with respect to the weighted input of the last layer $\frac{\partial C}{\partial \boldsymbol{z}^{(L-1)}}$, we can propagate gradient information backward in the computational graph to obtain all the gradients of this form. Then, we can use these quantities

to determine the gradients with regard to the model parameters at each layer according to the following equation:

$$\frac{\partial C}{\partial W_{ij}^{(l)}} = a_j^{(l-1)} \frac{\partial C}{\partial z_i^{(l)}} \tag{2.9}$$

This algorithm can be extended to arbitrary computational graphs provided that all forward computations are differentiable. It relies on automatic differentiation to evaluate the derivatives, and applies the chain rule to compute the model parameters gradients.

Especially, it can be extended to computational graphs corresponding to the temporal unfolding of RNNs. This extension, called Backpropagation Through Time (BPTT) (Werbos, 1988) is widely used to train RNNs.

As an example, we can apply BPTT on the sRNN model introduced before. The following paragraphs mostly rephrase the works of (Bengio et al., 1994; Pascanu et al., 2012; Pascanu et al., 2013) to provide an understanding of the issues that can arise when using BPTT to train RNNs. The recurrent weights $\boldsymbol{W}_r$ of the RNN intervene at each layer of the unrolled computational graph. If we suppose that C can be written as a sum of functions C_t depending on each output $\boldsymbol{x}_t$, we have:

$$\begin{aligned} \frac{\partial C}{\partial \boldsymbol{W}_r} &= \sum_{t=0}^{T-1} \frac{\partial C_t}{\partial \boldsymbol{W}_r} \\ &= \sum_{t=0}^{T-1} \sum_{k=0}^{t} \Big(\frac{\partial C_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} \frac{\partial^+ \boldsymbol{h}_k}{\partial \boldsymbol{W}_r} \Big) \end{aligned} \tag{2.10}$$

The gradient is developed into a double sum. The first sum over t allows covering the influence of $\boldsymbol{W}_r$ onto all the components C_t of the total loss function. The second sum allows covering the influence of $\boldsymbol{W}_r$ in the computations at each time step k preceding t. The notation $\frac{\partial^+ \boldsymbol{h}_k}{\partial \boldsymbol{W}_r}$ corresponds to the direct partial derivative of $\boldsymbol{h}_k$ with regard to $\boldsymbol{W}_r$ without using the chain rule to extend this derivative to past hidden states $\boldsymbol{h}_{j<k}$.

What is of interest to us in this equation is the term $\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k}$ that transports the gradient from time step t where the error signal originates, towards the previous time step k where we compute a component of the total gradient. This term can be further developed as:

$$\begin{aligned} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_k} &= \prod_{j=k+1}^{t} \frac{\partial \boldsymbol{h}_j}{\partial \boldsymbol{h}_{j-1}} \\ &= \prod_{j=k+1}^{t} \boldsymbol{J}_j \end{aligned} \tag{2.11}$$

where $\boldsymbol{J}_j$ is the Jacobian matrix of the RNN at time j. If the Jacobian matrices have large spectral radii, the long term components (where $k << t$) of the gradient tend to explode. This issue, called exploding gradient problem, can also arise for different reasons, that can be better understood using a dynamical system perspective on RNNs.

As explained previously, dynamical systems can contain attractors. These attractors are each associated with a part of the state space, called attractor basin, where a state converges to the attractor given enough time. It has been observed that small changes in the model parameters can lead to the appearance or disappearance of an attractor state in the dynamical system. For an initial state in the basin of the new attractor, the trajectory of the RNN state is now dramatically different than the trajectory using former model parameters. Another event of the same type is when a boundary between attractor basins is moved due to a change in model parameters. Again, an initial state that was in the first attractor basin now belongs to a new attractor basin, leading to dramatically different trajectories.

These types of event, sometimes called bifurcations (Doya, 1993), also complicate the training of RNNs compared to feedforward networks.

Finally, if we consider for instance the gradient of $\boldsymbol{W}_i$ with regard to a loss function depending only on $\boldsymbol{x}_{T-1}$, and with an input only available at time $t = 0$, this gradient only involves one long term component scaled by the product of all Jacobian matrices between $t = 1$ and $t = T - 1$. If the spectral radii of the Jacobian matrices are smaller than 1, this product vanishes to 0. This,

issue, called vanishing gradient, can arise when the network has to learn long term dependencies between inputs and outputs, as in our example.

A large variety of methods have been studied to mitigate those issues. Here we shortly review some of the existing approaches. To fix the exploding gradient problem, a candidate solution is to use a L1 or L2 penalty on the recurrent weights, as this tends to pull the spectral radius of the Jacobians towards values lower than 1. Another simple yet effective solution is to use gradient clipping, that is, renormalizing the gradient when it exceeds a chosen threshold.

To address the vanishing gradient problem, it has been suggested to augment RNN models with gating mechanisms (Hochreiter and Schmidhuber, 1997; Cho et al., 2014). This approach, as well as other methods engineering constraints on the RNN recurrent weights (Chang et al., 2019), is developed in section 3.2.2.

As the issues introduced here come from the learning of model parameters using gradient descent and BP, some approaches have tried to directly improve the learning algorithm, to use other learning methods, or to drastically work around the difficulties of learning the input and recurrent weights of RNNs.

A possible improvement to standard gradient descent is second order optimization. Hessian-free optimization (Martens, 2010; Martens and Sutskever, 2011) is a quasi-Newton technique that allows approaching second-order optimization without having to compute or approximate the Hessian matrix, and thus at a reasonable computational cost. This approach seems very efficient at dealing with the vanishing gradient problem.

Evolino (Schmidhuber et al., 2005; Schmidhuber et al., 2007) suggests to learn the recurrent weights using evolution strategies instead of gradient descent. Learning the output weights is a simpler optimization problem that can be solved without BP, for instance using linear regression.

Finally, the Reservoir Computing (RC) approach (Jaeger, 2001; Lukoševičius and Jaeger, 2009) completely dismiss the difficult question of learning recurrent weights by instead researching suitable initialization strategies, and performing learning only on the output layer. A more detailed description of this framework is provided in section 3.2.3.

2.2 Predictive coding

2.2.1 PC and the Bayesian brain

PC is a theory of brain function (Rao and Ballard, 1999; Clark, 2013) that has been very successful at explaining neurophysiological data, especially on low-level perception. It extends the idea that neural representations emerge as part of an inference process of the causes of sensory observations, as already introduced by Helmholtz in 1867 (Von Helmholtz, 1867). This so-called Bayesian brain hypothesis (Knill and Pouget, 2004) suggests that perception is a process of probabilistic inference, based on a generative (or predictive) model $p_\theta(\boldsymbol{x})$ of sensory observations $\boldsymbol{x}$, parameterized by a variable $\boldsymbol{\theta}$. As such, the mapping from low-level to high-level representations, $p_\theta(\boldsymbol{h}|\boldsymbol{x})$ results from an inversion, in the Bayesian sense, of the mapping from high-level to low-level representations (i.e. the generative model). An early implementation of these ideas is the Helmholtz machine (Dayan et al., 1995; Dayan and Hinton, 1996), which introduces a recognition model $q_\phi(\boldsymbol{h}|\boldsymbol{\theta})$ to perform approximate Bayesian inference, as represented in figure 2.5. This model is an unsupervised learning algorithm and is trained using the wake-sleep algorithm, that alternates between bottom-up update phases and top-down update phases. It can also be seen as a precursor of Variational Auto-Encoders (VAEs) (Kingma and Welling, 2014; Rezende et al., 2014; Doersch, 2021), that instead learn the parameters of the recognition and generative models using BP.

This combination of top-down generative, and bottom-up inference computations is at the core of PC. Contrary to Helmholtz machines or VAEs, the bottom-up computations in PC rely on a prediction error signal, denoted in this thesis by the greek letter $\boldsymbol{\epsilon}$. In Rao and Ballard's implementation of PC (Rao and Ballard, 1999), visual observations are predicted by a hierarchical generative model. Each layer of the hierarchy tries to predict the activation of the layer below. If the prediction is accurate, no inference of the above layer activation is needed. However, when there is a discrepancy between the prediction and actual value, the prediction error $\boldsymbol{\epsilon}$, defined as the difference between actual and predicted value, is processed by reciprocal connections in order to update the above layer activation. On the bottom layer, the actual activations are provided by external sensory inputs. The top-down prediction, and bottom-up inference computations all occur concurrently until the network converges to a stationary configuration. Intuitively, this

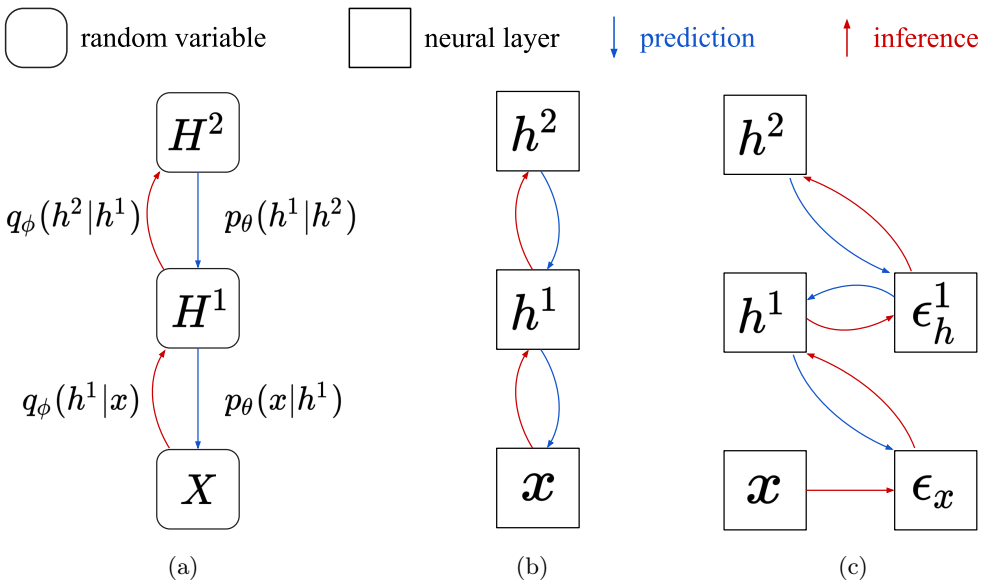


Figure 2.5: **Left:** Representation of the probabilistic hierarchical generative model according to the Bayesian brain hypothesis. In red is represented the recognition density that is used for approximate inference of the hidden variables H^1 and H^2 based on the observable variable X. **Center:** Neural network representation of a Helmholtz machine. The top-down computations perform prediction while the bottom-up computations perform inference. **Right:** Neural network representation of the PC approach. Neuron populations encoding prediction error at each layer are added into the neural network. They are denoted by the variables $\boldsymbol{\epsilon}_x$ and $\boldsymbol{\epsilon}_h$.

means that the information worth being transmitted up into the hierarchical representation is the mismatch between predicted and observed values. If the generative model and the configuration of layer activations provide a perfect prediction of sensory observations, then the inference process no longer needs to update the latent representation. We can say that the inferred representation explains away the incoming sensory observation.

The described approach is illustrated in figure 2.5c. At each layer, there is a population encoding the agent's representation about its sensory input, as well as a parallel population encoding the error between the representation and the prediction coming from the upper layer.

Presented as such, the connection with the Bayesian brain hypothesis and variational inference is far from obvious, since at first glance, only deterministic (in opposition to probabilistic) variables are processed in this PC network. In the next section, we introduce the free-energy principle and show how applying this principle to perception reconciles variational inference and PC. In section 2.2.3, we derive a learning algorithm based on PC and in 2.2.4 we review some neurophysiological experiments and provide discussions about PC as a candidate model of the brain. Finally in 2.2.5 we review existing implementations more or less closely related to PC.

2.2.2 Free-energy formulation of PC

The Free Energy Principle (FEP) (Friston, 2009; Friston, 2010b) is an ambitious framework aiming at unifying cognition under the common principle of surprise minimization. The goal of this section is to highlight the connection between the FEP and PC. Starting from the FEP hypotheses, and relying on a few simplifications, we derive an algorithm that can be translated as neural network dynamics aligning with the PC theory.

In this section, we adapt some derivations from the very useful mathematical reviews of the FEP presented in (Bogacz, 2017) and (Buckley et al., 2017). These articles provide a technical guide of the FEP mathematical framework by compiling a history of publications on the subject (Friston, 2003; Friston, 2005; Friston and Kilner, 2006; Friston et al., 2007; Friston, 2008a; Friston et al., 2008; Friston, 2009; Friston and Kiebel, 2009; Friston, 2010a). A part of these derivations are available in appendix A. In the main text we only present the key equations and the assumptions needed to simplify the expression of Variational Free-Energy (VFE). The sequence memory models

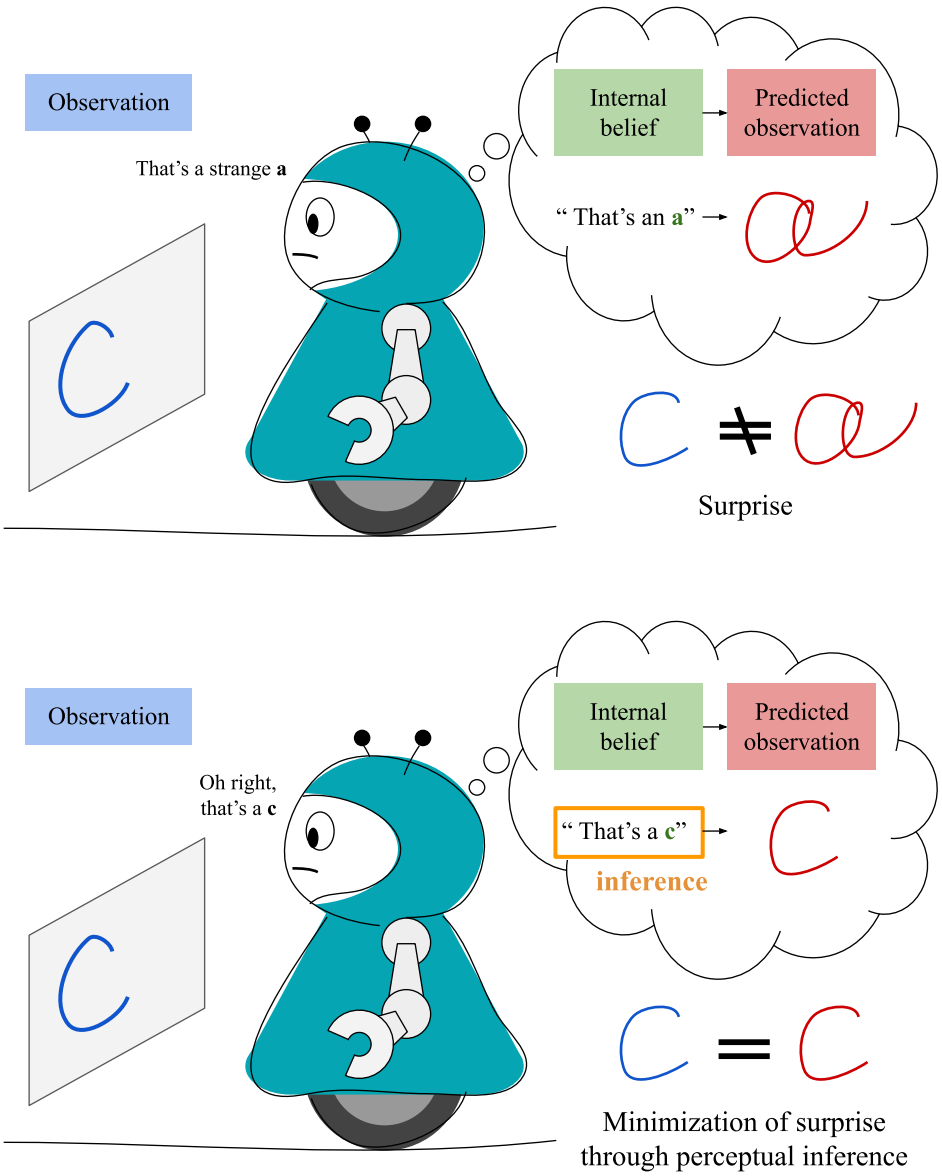


Figure 2.6: Illustration of surprise minimization through perceptual inference.

proposed in this thesis all build upon the derivations shown in this section.

Starting from the idea that biological organisms need to be in homeostasis with their environment, the FEP suggests minimization of surprise as a necessary mechanism for survival. Surprise, in our information-theoretic definition, corresponds to the negative log probability of sensory observations $\boldsymbol{x}$ according to a predictive (or generative) model.

$$\mathcal{S}(\boldsymbol{x}) = -\log p(\boldsymbol{x}) \tag{2.12}$$

According to the FEP, living agents are equipped with such generative models, and cognitive functions such as perception, learning, and action, can be explained as processes of surprise minimization. Figure 2.6 illustrates the idea of surprise minimization through perceptual inference. In this example, the cognitive agent first has a prior belief of seeing the letter *a*. Its internal model generates a sensory prediction. Since this sensory prediction does not match its observation, the cognitive agent experiences surprise. To minimize this surprise, it can update its internal belief so that its new prediction matches its sensory observation. In our example, the cognitive agent updates its internal belief to that of seeing the letter *c*, thus resolving the prediction error and

minimizing surprise.

According to the FEP, surprise minimization is performed through the minimization of VFE. VFE, denoted F in our equations, is a quantity equivalent to the negative Evidence Lower Bound (ELBO) often used in variational Bayesian methods. It is defined as:

$$F(\boldsymbol{x}) = \int_h \log\Big(\frac{q(\boldsymbol{h})}{p(\boldsymbol{x},\boldsymbol{h})}\Big) q(\boldsymbol{h}) d\boldsymbol{h} \tag{2.13}$$

We can show that this quantity constitutes an upper bound on surprise, and thus can act as a proxy for surprise minimization:

$$\begin{aligned} F(\boldsymbol{x}) &= \int_h \log\Big(\frac{q(\boldsymbol{h})}{p(\boldsymbol{x},\boldsymbol{h})}\Big) q(\boldsymbol{h}) d\boldsymbol{h} && (2.14)\\ &= \int_h \log\Big(\frac{q(\boldsymbol{h})}{p(\boldsymbol{h}|\boldsymbol{x})}\Big) q(\boldsymbol{h}) d\boldsymbol{h} - \int_h \log p(\boldsymbol{x}) q(\boldsymbol{h}) d\boldsymbol{h} && (2.15)\\ &= D_{KL}\big(q(\boldsymbol{h}) \| p(\boldsymbol{h}|\boldsymbol{x})\big) + \mathcal{S}(\boldsymbol{x}) && (2.16) \end{aligned}$$

where $\boldsymbol{h}$ denotes the hidden latent variable of the generative model, and D_{KL} denotes the Kullback-Leibler divergence. Since the Kullback-Leibler divergence is non-negative, surprise is upper bounded by the VFE. These quantities become equal if the divergence reaches 0, that is, when the distribution q is equal to the posterior distribution $p(\boldsymbol{h}|\boldsymbol{x})$. For this reason, we might sometimes call q the approximate posterior density instead of recognition density. Since VFE is an upper bound on surprise, minimizing VFE also minimizes surprise.

This rewriting of VFE is useful to prove its connection to surprise, but it does not reflect the major interest of variational inference compared to direct Bayesian inference. Indeed, Bayesian inference suffers from computational intractability as we need to sum over all possible states $\boldsymbol{h}$ to estimate the probability $p(\boldsymbol{x}) = \int p(\boldsymbol{x}|\boldsymbol{h}) p(\boldsymbol{h}) d\boldsymbol{h}$. In contrast, if we constraint the distribution q to be non-null on a small set of values (or tightly shaped around a small number of values), integrating on $q(\boldsymbol{h})$ becomes tractable (or approximable). Computing and optimizing VFE is thus interesting when it does not rely on quantities such as $p(\boldsymbol{x})$ or $p(\boldsymbol{h}|\boldsymbol{x})$. A first interesting rewriting of VFE is:

$$F(\boldsymbol{x}) = \underbrace{D_{KL}\big(q(\boldsymbol{h}) \| p(\boldsymbol{h})\big)}_{\text{Complexity}} - \underbrace{\mathbb{E}_q\big[p(\boldsymbol{x}|\boldsymbol{h})\big]}_{\text{Accuracy}} \tag{2.17}$$

where $\mathbb{E}_q$ denotes the expectation with regard to the density q. The first quantity of this equation is called *complexity*. Intuitively, it scores how complex the recognition density is compared to the prior density. The second quantity is called *accuracy*, and measures how good the recognition density q is at predicting the observed variable $\boldsymbol{x}$. This expression of VFE corresponds (up to a change of sign) to the usual expression of the ELBO loss function often used in variational Bayes methods. As our goal here is to show how variational inference aligns with PC, we need to start from another, equivalent way of writing down VFE:

$$F(\boldsymbol{x}) = \underbrace{\int_h E(\boldsymbol{x},\boldsymbol{h}) q(\boldsymbol{h}) d\boldsymbol{h}}_{\text{Expected energy}} + \underbrace{\int_h \log\big(q(\boldsymbol{h})\big) q(\boldsymbol{h}) d\boldsymbol{h}}_{\text{(Negative) entropy}} \tag{2.18}$$

where

$$E(\boldsymbol{x},\boldsymbol{h}) = -\log p(\boldsymbol{x},\boldsymbol{h}) \tag{2.19}$$

is called *energy*. The first term of this equation thus corresponds the expectation of energy with regard to the recognition density q. The second term is the entropy of the recognition density q.

In the FEP framework, perceptual inference is framed as a process of free-energy minimization by optimizing the recognition density q. In practice, q is constrained to certain classes of probability distributions to simplify the optimization problem. Here we assume that the recognition density takes a Gaussian form, and VFE is minimized by varying the parameters of this distribution.

$$q(\boldsymbol{h}) = \frac{1}{\sqrt{(2\pi)^{d_h}|\boldsymbol{\Sigma}|}} \exp\Big(-\frac{1}{2}(\boldsymbol{h}-\boldsymbol{m}_h)^\intercal \cdot \boldsymbol{\Sigma}^{-1} \cdot (\boldsymbol{h}-\boldsymbol{m}_h)\Big) \tag{2.20}$$

To connect this equation to neural dynamics aligning with the PC theory, more assumptions are needed:

- First, we assume that the recognition distribution $q(\boldsymbol{h})$ is tightly shaped around the mean $\boldsymbol{m}_h$. This allows us to approximate the intractable integrals in equation 2.18.
- Second, we assume that the generative model $p(\boldsymbol{x}, \boldsymbol{h})$ is hierarchical and made of a cascade of multivariate Gaussian distributions. The Gaussian distribution $p(\boldsymbol{h}^{(i)}|\boldsymbol{h}^{(i+1)})$ at layer $i \in [0, l-1]$ is characterized by its mean vector $\boldsymbol{g}^{(i+1)}(\boldsymbol{h}^{(i+1)})$ and covariance matrix $\boldsymbol{\Sigma}_i$. The functions $\boldsymbol{g}^{(i)}$ depend on variables $\boldsymbol{\theta}^{(i)}$. The distribution on the top layer l is characterized by its mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}_l$. The mean vector $\boldsymbol{\mu}$, the covariance matrices $\boldsymbol{\Sigma}_i$ and the variables $\boldsymbol{\theta}_i$ are the parameters of the generative model.
- Third, we assume that the covariance matrix at each layer of the hierarchical generative model is proportional to the identity matrix:

  $$\boldsymbol{\Sigma_i} = \sigma_i^2 \mathbb{I}_{d_{h^{(i)}}}$$

  We discuss a less constraining assumption in section 3.3.6.1.
- Finally, we define the prediction errors $\boldsymbol{\epsilon}_i$ at each layer as the difference between the the prediction coming from the upper layer and the current recognition density mean:

  $$\begin{aligned} \boldsymbol{\epsilon}^{(0)} &= x - \boldsymbol{g}^{(1)}(\boldsymbol{m}_h^{(1)}) \\ \boldsymbol{\epsilon}^{(i)} &= \boldsymbol{m}_h^{(i)} - \boldsymbol{g}^{(i+1)}(\boldsymbol{m}_h^{(i+1)}) \\ \boldsymbol{\epsilon}^{(l)} &= \boldsymbol{m}_h^{(l)} - \boldsymbol{\mu} \end{aligned} \tag{2.21}$$

  The definition of prediction errors has the advantage of simplifying the expression of the VFE, but also are meaningful when transcribing our inference algorithm into a neural network architecture.

Using all these assumptions, we obtain the following expression for the VFE. The detailed derivations of this equation are provided in appendix A.

$$F(\boldsymbol{x}, \boldsymbol{m}_h^{(1)}, \dots, \boldsymbol{m}_h^{(l)}) = \frac{1}{2} \sum_{i=0}^{l} \frac{1}{\sigma_i^2} \boldsymbol{\epsilon}^{(i)\intercal} \cdot \boldsymbol{\epsilon}^{(i)} + C' \tag{2.22}$$

where C' is a constant. Perceptual inference corresponds to the optimization of the recognition density means $\boldsymbol{m}_h^{(i)}$ in order to minimize the VFE as described above. The FEP suggests that neural dynamics implement an iterative gradient descent minimization of this quantity, using two neural populations at each layer. The first neural population encodes at each layer the recognition density mean $\boldsymbol{m}_h^{(i)}$, and the second neural population encodes at each layer the prediction error $\boldsymbol{\epsilon}^{(i)}$.

Applying gradient descent to the recognition density means with regard to the VFE entails the following update rule:

$$\boldsymbol{m}_h^{(i)} \leftarrow \boldsymbol{m}_h^{(i)} - \alpha \nabla_{\boldsymbol{m}_h^{(i)}} F \tag{2.23}$$

$$\leftarrow \boldsymbol{m}_h^{(i)} + \underbrace{\frac{\alpha}{\sigma_{i-1}^2} \boldsymbol{g}^{(i)'}(\boldsymbol{m}_h^{(i)}) \cdot \boldsymbol{\epsilon}_{i-1}}_{\text{Bottom-up}} \underbrace{- \frac{\alpha}{\sigma_i^2} \boldsymbol{\epsilon}_i}_{\text{Top-down}} \tag{2.24}$$

where α is a coefficient that determines the rate of the iterative update process. If the generative model is composed of a cascade of linear layers with some activation function $\boldsymbol{f}$, parameterized by weights $\boldsymbol{W}_i$, we can write down the derivative of $\boldsymbol{g}^{(i)}$ and get:

$$\boldsymbol{m}_h^{(i)} \leftarrow \boldsymbol{m}_h^{(i)} + \frac{\alpha}{\sigma_{i-1}^2} \boldsymbol{f}'(\boldsymbol{m}_h^{(i)}) \odot \left(\boldsymbol{W}_i^\intercal \cdot \boldsymbol{\epsilon}_{i-1} \right) - \frac{\alpha}{\sigma_i^2} \boldsymbol{\epsilon}_i \tag{2.25}$$

We have an update rule for the recognition density mean $\boldsymbol{m}_h^{(i)}$ that only depends on local information, that is the prediction error of the very layer i, and the prediction error on the lower layer $i-1$. As we have seen before in equation 2.21, the prediction error at a layer i only depends on the recognition density mean on this layer and the recognition density mean of the upper layer. Intuitively, the first component pulls it into a direction that minimizes the prediction error on the lower layer, while the second component pulls it into a direction that minimizes the prediction error on the current layer. This cascade of local iterative optimizations transports information about the prior $\boldsymbol{\mu}$ in a top-down fashion, and information about the observed variable $\boldsymbol{x}$ in a bottom-up fashion, until a stationary configuration is reached.

Based on this update rule, we can design an inference algorithm that iteratively updates recognition density means and prediction errors until convergence. We now use a subscript t to denote the temporal dynamics of the variables $\boldsymbol{m}_h^{(i)}$ and $\boldsymbol{\epsilon}_i$. The perceptual inference algorithm is presented in algorithm 1.

Algorithm 1: A PC algorithm for perceptual inference

Parameters: $\boldsymbol{\mu}, \{\sigma_0^2, \dots, \sigma_l^2\}, \{\boldsymbol{W}_1, \dots, \boldsymbol{W}_l\}, \alpha$
Input: $\boldsymbol{x}$
Initialize $\{\boldsymbol{m}_{h,0}^{(1)}, \dots, \boldsymbol{m}_{h,0}^{(l)}\}$;
Initialize $\{\boldsymbol{\epsilon}_{0,0}, \dots, \boldsymbol{\epsilon}_{l,0}\}$;
for $0 \leq t < T$ **do**
  $\boldsymbol{\epsilon}_{0,t+1} \leftarrow \boldsymbol{x} - \boldsymbol{f}(\boldsymbol{W}_1^\intercal \cdot \boldsymbol{m}_{h,t}^{(1)})$;
  for $1 \leq i < l$ **do**
    $\boldsymbol{m}_{h,t+1}^{(i)} \leftarrow \boldsymbol{m}_{h,t}^{(i)} + \frac{\alpha}{\sigma_{i-1}^2} \boldsymbol{f}'(\boldsymbol{m}_{h,t}^{(i)}) \odot \left(\boldsymbol{W}_i^\intercal \cdot \boldsymbol{\epsilon}_{i-1,t}\right) - \frac{\alpha}{\sigma_i^2} \boldsymbol{\epsilon}_{i,t}$;
    $\boldsymbol{\epsilon}_{i,t+1} \leftarrow \boldsymbol{m}_{h,t}^{(i)} - \boldsymbol{f}(\boldsymbol{W}_{i+1}^\intercal \cdot \boldsymbol{m}_{h,t}^{(i+1)})$;
  end
  $\boldsymbol{m}_{h,t+1}^{(l)} \leftarrow \boldsymbol{m}_{h,t}^{(l)} + \frac{\alpha}{\sigma_{l-1}^2} \boldsymbol{f}'(\boldsymbol{m}_{h,t}^{(l)}) \odot \left(\boldsymbol{W}_l^\intercal \cdot \boldsymbol{\epsilon}_{l-1,t}\right) - \frac{\alpha}{\sigma_l^2} \boldsymbol{\epsilon}_{l,t}$;
  $\boldsymbol{\epsilon}_{l,t+1} \leftarrow \boldsymbol{m}_{h,t}^{(l)} - \boldsymbol{\mu}$;
end

This algorithm iteratively updates the activation of neural populations encoding the current beliefs about latent states, and the activation of neural populations encoding prediction errors. It describes a computational graph that can also be obtained based on a hierarchical RNN similar to the network represented in figure 2.5c. This neural network and the underlying computations align precisely with the PC theory. We can thus conclude that PC can be seen as perceptual inference performed using variational Bayesian methods, and additional assumptions on the generative and recognition densities.

To illustrate this algorithm in action, we have implemented the PC-network displayed in figure 2.5c. This recurrent network is based on a two-layered perceptron to which we add two parallel populations of neurons encoding prediction error: one for the output layer, and one for the hidden layer. It is represented in figure 2.7. We use a toy data set of 4 visual patterns composed of 20 pixels, representing the letters A, B, C and D. We train the network to predict the k-th visual pattern when setting the top layer to the one-hot vector activated on the k-th neuron. The learning algorithm is described in the next section. After training, we provide the pattern C as observed output $\boldsymbol{x}$. The coefficients $\frac{\alpha}{\sigma_0^2}$ and $\frac{\alpha}{\sigma_1^2}$ are set to 0.2. We do not consider the influence of the prior mean $\boldsymbol{\mu}$, which is equivalent to choosing a very high value for σ_2 (the Gaussian prior becomes flat). We display the evolution of $\boldsymbol{m}_h^{(1)}$, $\boldsymbol{m}_h^{(2)}$ and the predicted observation during 15 inference steps.

We can see that the inference process quickly converges to a configuration of its latent states that properly predicts the observed visual pattern C. Additionally, we can see that the top layer has converged to a representation very close to the one-hot vector activated on the neuron corresponding to the pattern C. We can conclude that this perceptual inference process performs a form of classification of the provided observation. These results were obtained with a hidden state dimensions $d_h^{(1)} = 10$ and $d_h^{(2)} = 4$. The inference is parametered by the coefficients α, σ_0^2 and σ_1^2 that were set to $\alpha = 0.2$, $\sigma_0^2 = 1$ and $\sigma_1^2 = 1$. The source code used for this experiment is available

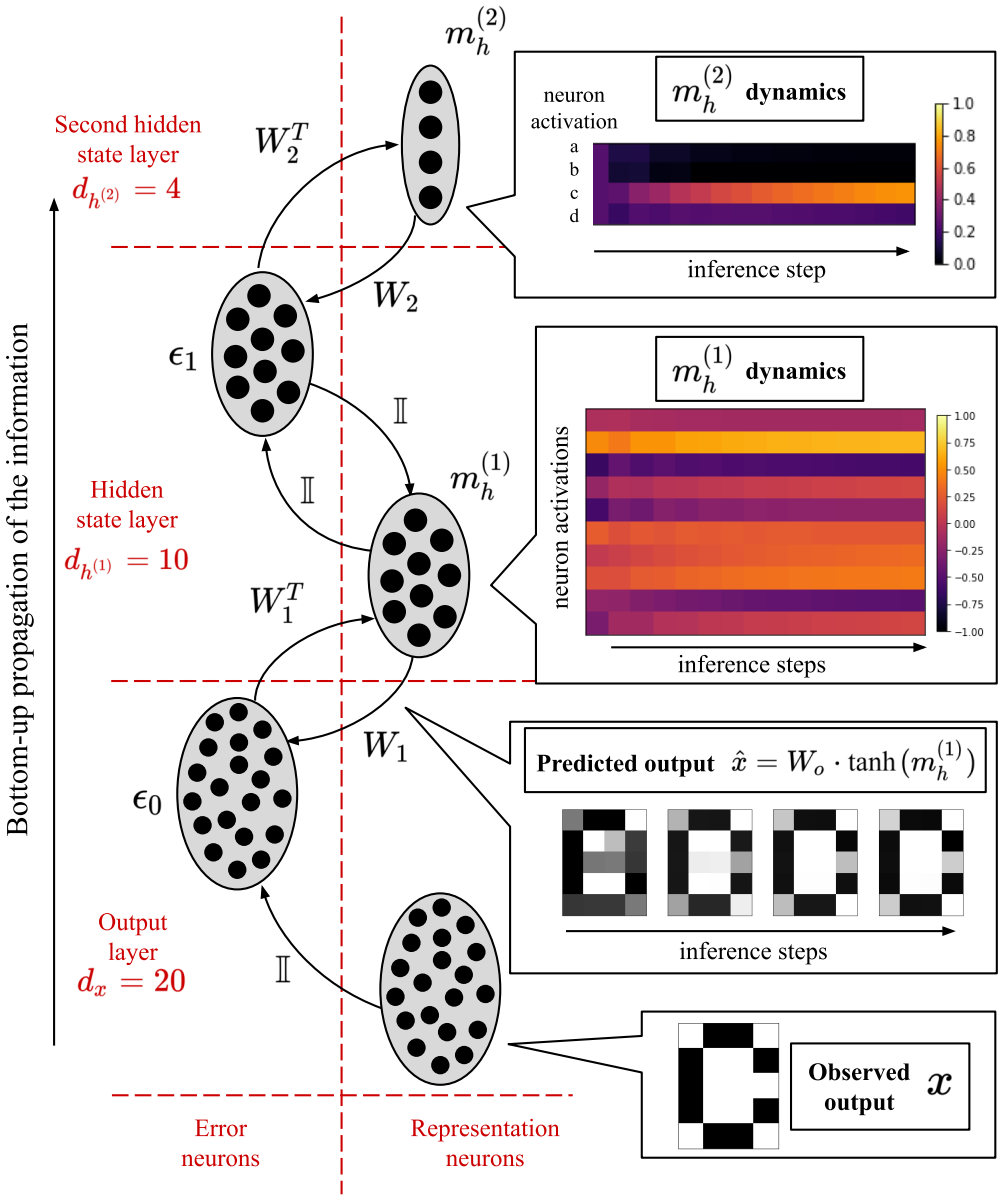


Figure 2.7: Illustration of the PC inference algorithm on a toy example.

on GitHub[1].

To conclude this section, we highlight that the algorithm we have derived from the FEP does not generalize all models and algorithms that have been proposed in the PC literature. To be more precise, we have shown how the FEP applied to a certain generative model, with some additional assumptions on the recognition density, can describe an algorithm that implements PC. In the remaining of this thesis, we design models that fall in the intersection of the PC and FEP theories, building upon the former derivations.

2.2.3 Learning

We have seen how PC describes perception as an iterative inference process that can be performed in an RNN, using only local update rules. Here we show how learning can be performed in such a neural architecture reusing the FEP framework developed in the previous section. Furthermore, we present some proof (Whittington and Bogacz, 2017; Millidge et al., 2020a) of how these learning mechanisms approximate the BP algorithm introduced previously.

Similarly to perceptual inference, learning in the FEP is framed as a process of minimization

[1]https://github.com/sino7/example_pc_network

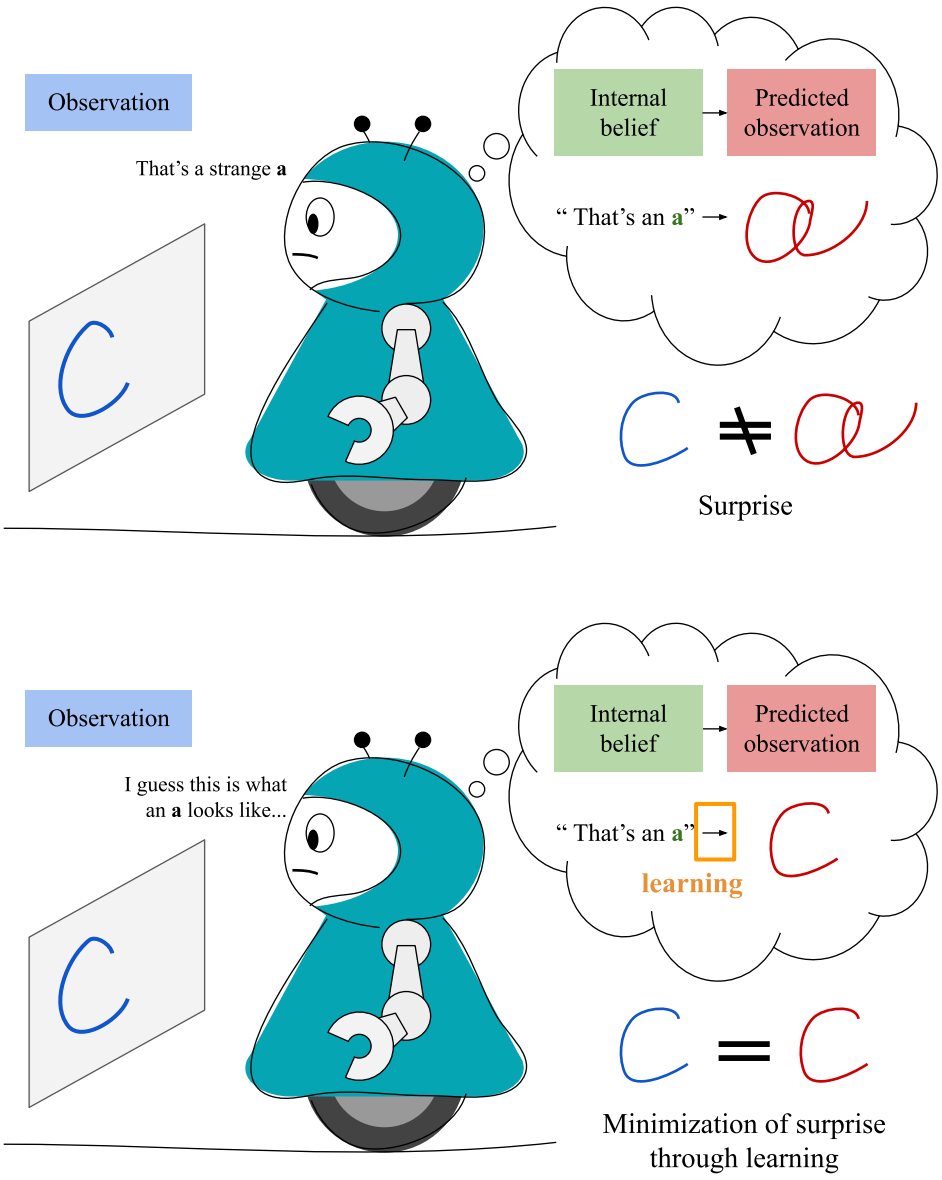


Figure 2.8: Illustration of surprise minimization through learning.

of VFE, typically operating at a slower pace. If inference corresponds to the process of updating the recognition density parameters, learning refers to the process of optimizing the parameters of the generative model, such as the collection of covariance matrices $\boldsymbol{\Sigma}_i$, and the parameters of the functions $\boldsymbol{g}^{(i)}$ that we denote $\boldsymbol{\theta}_i$ (for the general case). This idea is illustrated in figure 2.8. In this picture, the cognitive agent updates its generative model instead of modifying its internal belief. After learning, the generative model predicts that the internal belief a corresponds to the visual input of a c. This learning process minimizes surprise, since after learning the agent's prediction matches its sensory observation.

If we consider the hierarchical neural network described in the previous section, where each layer tries to predict the activation of the lower layer through a linear mapping, the parameters $\boldsymbol{\theta}_i$ of the functions $\boldsymbol{g}^{(i)}$ correspond to synaptic weights. Learning as an optimization of the generative model parameters thus aligns with the classical notion of learning synaptic weights in neural networks. The adaptation of the covariance matrices, or of their inverses, precision matrices, is discussed in section 3.3.6.1.

The model parameters $\boldsymbol{\theta}_i$ are updated following a gradient descent on the VFE using a learning

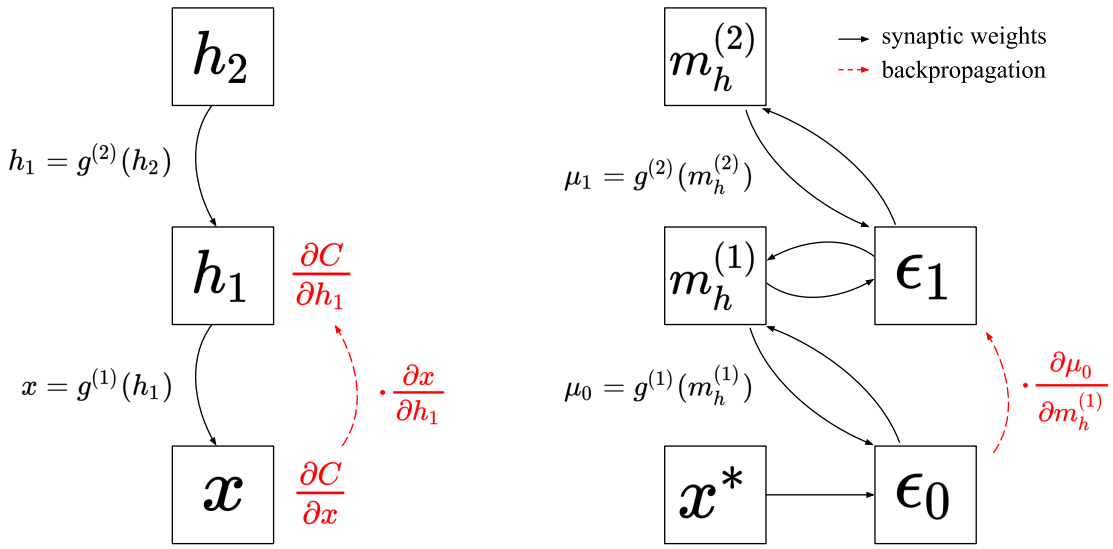


Figure 2.9: Illustration of the parallelism between BP and the PC learning algorithm. Adapted from (Millidge et al., 2020a).

rate λ:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \lambda \frac{\partial F}{\partial \boldsymbol{\theta}_i} \tag{2.26}$$

We start from the expression of the VFE provided in equation 2.22, and assume that covariance matrices are proportional to the identity, and identical across layers. Additionally, we introduce the notation $\boldsymbol{\mu}_i = \boldsymbol{g}^{(i+1)}(\boldsymbol{m}_h^{(i+1)}; \boldsymbol{\theta}_{i+1})$ to denote the prediction of the i-th layer latent variable originating from layer $(i+1)$.

$$\begin{aligned} \frac{\partial F}{\partial \boldsymbol{\theta}_i} &= \frac{\partial F}{\partial \boldsymbol{\mu}_{i-1}} \frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{\theta}_i} \\ &= -\frac{1}{\sigma^2} \boldsymbol{\epsilon}_{i-1}^\intercal \cdot \frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{\theta}_i} \end{aligned} \tag{2.27}$$

which entails the following learning rule:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \frac{\lambda}{\sigma^2} \boldsymbol{\epsilon}_{i-1}^\intercal \cdot \frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{\theta}_i} \tag{2.28}$$

Interestingly, this learning rule only involves local information at each layer. The parameters $\boldsymbol{\theta}_i$ are pulled in a direction that minimizes the prediction error on the lower layer $\boldsymbol{\epsilon}_{i-1}$. If we consider the case where predictions $\boldsymbol{\mu}_{i-1}$ are computed as a linear mapping of the input $\boldsymbol{m}_h^{(i)}$, with a weight matrix $\boldsymbol{\theta}_i$, then the learning rule becomes:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \frac{\lambda}{\sigma^2} \boldsymbol{\epsilon}_{i-1}^\intercal \cdot \boldsymbol{m}_h^{(i)} \tag{2.29}$$

With this additional assumption, the update takes the form of a Hebbian learning rule, since $\boldsymbol{\epsilon}_{i-1}$ and $\boldsymbol{m}_h^{(i)}$ correspond to the activations of the presynaptic and postsynaptic layers.

Without relying on this assumption, a surprising property of the inference and learning rules of the presented PC model is that if inference reaches a stationary configuration before applying the learning rule, the learning rule exactly replicates the parameter updates of BP. We can verify this by first looking at the relationship between prediction error at different layers when a stationary configuration is reached:

$$\frac{d\boldsymbol{m}_h^{(i)}}{dt} = -\alpha \frac{\partial F}{\partial \boldsymbol{m}_h^{(i)}} = 0 \tag{2.30}$$

And consequently:

$$\frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{m}_h^{(i)}} \cdot \boldsymbol{\epsilon}_{i-1}^* - \boldsymbol{\epsilon}_i^* = 0 \tag{2.31}$$

$$\boldsymbol{\epsilon}_i^* = \frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{m}_h^{(i)}} \cdot \boldsymbol{\epsilon}_{i-1}^* \tag{2.32}$$

On the other hand, if we were to apply BP on the feedforward neural network embedding the generative model represented in figure 2.9, we would obtain the following equation that transports gradient information back into the hierarchy:

$$\frac{\partial C}{\partial \boldsymbol{h}_i} = \frac{\partial \boldsymbol{h}_{i-1}}{\partial \boldsymbol{h}_i} \cdot \frac{\partial C}{\partial \boldsymbol{h}_{i-1}} \tag{2.33}$$

If we compare both models, we can notice that the quantities $\frac{\partial \boldsymbol{\mu}_{i-1}}{\partial \boldsymbol{m}_h^{(i)}}$ in the first model and $\frac{\partial \boldsymbol{h}_{i-1}}{\partial \boldsymbol{h}_i}$ in the second model are equivalent. Consequently, the equations 2.32 and 2.33 yield exactly the same recurrence relation between respectively prediction errors, and gradients, between layers i and $i-1$.

Moreover, at the bottom layer, if the cost function used is the classical L2 norm, we can see that:

$$\frac{\partial C}{\partial \boldsymbol{h}_0} = \frac{\partial C}{\partial \boldsymbol{x}} = \frac{\partial \|\boldsymbol{x}^* - \boldsymbol{x}\|^2}{\partial \boldsymbol{x}} = -2(\boldsymbol{x}^* - \boldsymbol{x}) = -2\boldsymbol{\epsilon}_0^* \tag{2.34}$$

On the bottom layer, the gradient $\frac{\partial C}{\partial \boldsymbol{h}_0}$ and the prediction error $\boldsymbol{\epsilon}_0^*$ are proportional. Additionally, both quantities are related to the upper gradients and prediction errors using the same recurrence relation. Consequently, according to the principle of induction, we can conclude that for all i:

$$\boldsymbol{\epsilon}_i^* = -\frac{1}{2}\frac{\partial C}{\partial \boldsymbol{h}_i} \tag{2.35}$$

Finally, the learning rules prescribed by BP and by the gradient descent on the VFE both exploit this quantity in the same fashion:

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i - \lambda \frac{\partial C}{\partial \boldsymbol{h}_i}^\intercal \cdot \frac{\partial \boldsymbol{h}_i}{\partial \boldsymbol{\theta}_i} \tag{2.36}$$

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_i + \frac{\lambda}{\sigma^2} \boldsymbol{\epsilon}_i^{*\intercal} \cdot \frac{\partial \boldsymbol{\mu}_i}{\partial \boldsymbol{\theta}_i} \tag{2.37}$$

We can conclude that on the condition that we let the PC inference mechanism reach a stationary configuration, the PC learning rule is equivalent to the learning rule prescribed by BP using a L2 objective function. These results have been experimentally verified and generalized to arbitrary computational graphs in (Whittington and Bogacz, 2017; Millidge et al., 2020a).

We have seen that to reach the same learning algorithm, we have assumed that the generative model variances σ_i^2 were equal at each layer. Interestingly, by relaxing this simplification, the backward propagation algorithm would be weighted at each layer by the ratio between the variances of layers i and $i+1$. This additional factor might help mitigate the vanishing or exploding gradient issues observed with BP. It could be interesting to investigate the impact of precision (inverse variance) coefficients learning onto the prediction error propagation mechanism.

In practice, it is not necessary to perform inference until convergence of the neural activations before applying the learning rule. The time allocated for inference might depend on external constraints or can be bounded to limit the computational cost.

2.2.4 PC as a model of brain function

The necessity of running the inference algorithm for a large number of iterations before performing one learning update makes this learning algorithm way less efficient than BP. However, the inference and learning algorithms based on PC might be more biologically plausible compared to BP and BPTT. Here we shortly discuss some issues regarding the plausibility of an implementation of BP in the brain, and how the proposed algorithm based on PC might solve them:

- The first issue with BP is that layers should in some way be able to communicate their derivatives for gradient computations. Using the PC-based learning algorithm presented before does not solve this issue, as the derivatives of activation functions are still needed for bottom-up updates.
- The second issue we can raise is that the connections communicating gradient backwards need to have the same weights as the forward connections. This weight transport (Crick, 1989) is not possible in the brain, as it would need a perfect copy of forward synapses for backward computations. The PC based learning algorithm does not solve this issue as it relies on transposed forward weights for bottom-up updates.
- Finally, BP supposes that there exists a signal propagated backward that only affects the synaptic weights without impacting the neural activations, a mechanism with no biological correlate. On the contrary, combining bottom-up inference with local synaptic weights adaptation, as is done in the presented PC based learning algorithm, constitutes a more biologically plausible mechanism.

Although the presented PC based learning algorithm does not solve the first and second issues regarding biological plausibility, it can be modified to account for those. Both remaining issues come from the fact that we have forced the bottom-up connections to perform a local gradient descent on free-energy, thus requiring weight transport (copy of the top-down weights) and knowledge of the derivatives of activation functions. Instead, we can consider a backward circuitry with standard activation functions and synaptic weights that are either random (Lillicrap et al., 2016), or learned according to different local update rules (Rao and Ballard, 1997; Ororbia et al., 2020; Millidge et al., 2020b).

The PC architecture we presented also has some additional implausibilities due to its structure. The one-to-one connection pattern between error neurons and prediction neurons is unlikely to occur naturally in the brain. To solve this issue, (Millidge et al., 2020b) suggest learning these connections as well, although this modification seems to affect the model's overall performance.

In our derivations, we have seen how to apply the PC learning algorithm simple hierarchical computational graph. In fact, this method can be applied to any computational graph, including the computational graphs obtained by temporally unfolding RNNs. For RNNs, using the described inference mechanism to adjust past layer activations is not biologically plausible as these quantities have evolved in the meantime because of the recurrent connections. In chapter 3, we see how the FEP extends to temporal signals and we derive RNN models implementing free-energy minimization without the need of propagating prediction errors into the past.

Sticking to the static case, we have seen that PC can entail biologically plausible learning mechanisms. This adds to the ability of the PC theory to explain a large panel of neurophysiological and psychophysical results in visual and auditory systems, especially the learning of Gabor-like receptive fields (Rao and Ballard, 1999; Hosoya et al., 2005), top-down modulation of perceptual processing (Fenske et al., 2006; Summerfield and Koechlin, 2008; Summerfield et al., 2008), response damping with stimulus predictability (Müller et al., 1999; Alink et al., 2010), biphasic responses (Jehee and Ballard, 2009), mismatch negativity (Hughes et al., 2001; Rinne et al., 2005; Ouden et al., 2008; Stefanics et al., 2014), and the joint effect of top-down predictions and attention (Schröger et al., 2015; Smout et al., 2019).

Let us suppose for a moment that biological systems indeed implement the PC-based learning mechanisms we have described. We can question why evolutionary mechanisms have converged towards this solution. First, this solution may be far from optimal. After all, evolution implements an optimization method that can easily be stuck in local optima. Second, this solution may be indeed optimal considering the biological hardware at hand. BP might be a better learning algorithm than PC-based methods, but it cannot be implemented using neurons. PC-based methods could be the best approximation of BP that evolution has reached with these biological constraints. Finally, it is also possible that PC-based learning brings a significant selective advantage compared with BP that we have not yet discovered.

For instance, PC-based learning could have better energy efficiency than possible implementations of BP in the brain. This idea could encourage us to develop neuromorphic hardware especially mimicking the PC organization of neural processing units to reduce the energy consumption of ANN systems.

Even without dedicated hardware, the PC-based learning algorithm could scale better to very

deep networks than BP in terms of computation time. Indeed, if we look at the BP algorithm, we can see that the update rule for a layer k needs to await for the layer $k-1$ to finish its computations. This translates into a full forward propagation followed by a full backward propagation of information. After having transmitted forward its activation to the layer $k-1$, the layer k remains idle until the BP reaches it. On the contrary, in the PC algorithm, all layers are active at each point in time, always adjusting their activations and weight parameters in order to locally minimize VFE. Still, the fact that PC-based update rules only properly approximate BP if enough time is given for the inference process to stabilize could counter the relative interest of parallel computing in PC-based models.

2.2.5 Variants of the PC architecture

In this section, we review models related to the PC theory. This listing does not aim at being exhaustive, since this field of research has been very active for the last two decades. Instead, we aim at displaying key mechanisms for inference and learning using error propagation, and their variations along different models presented in the literature.

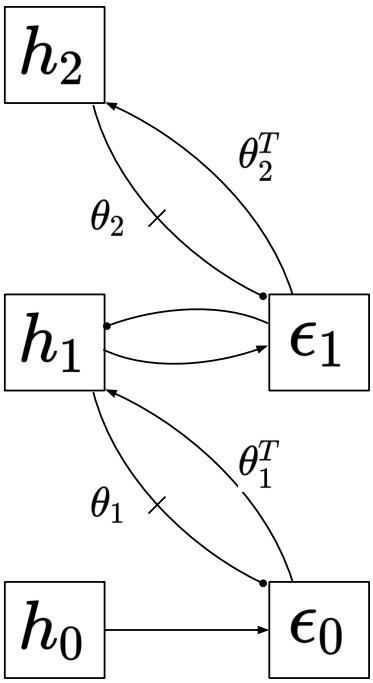
The PC model proposed in (Rao and Ballard, 1999) is at the basis of the PC theory. The neural network architecture, represented in figure 2.10a, approximates Bayesian inference in a way that is very close to the model we derived from the FEP. The hierarchical architecture comprises a population of representation neurons and a population of prediction error neurons at each layer. The representation neurons have top-down synaptic connections towards the lower layer prediction error neurons. Symmetrically, the error neurons of the lower layer have bottom-up synaptic connections towards the upper representation neurons, that correspond to a transposed copy of the top-down weights. The connections between representation and prediction error neurons on the same layer implement a one-to-one mapping. Learning occurs according to the same update rules we have described. This model constitutes an entry point for the comparison with all the models we present in this section.

In (Spratling, 2008; Spratling et al., 2009), the authors develop PC models using Divisive Input Modulation (DIM). This technique formulates a divisive version of prediction error, as the element-wise division of targets by predictions (a small positive term is added to the prediction to avoid division by 0). Using this error signal, they build inference and weight update rules that relate the method to non-negative matrix factorization. They show that their algorithm aligns with biased competition models of the brain.

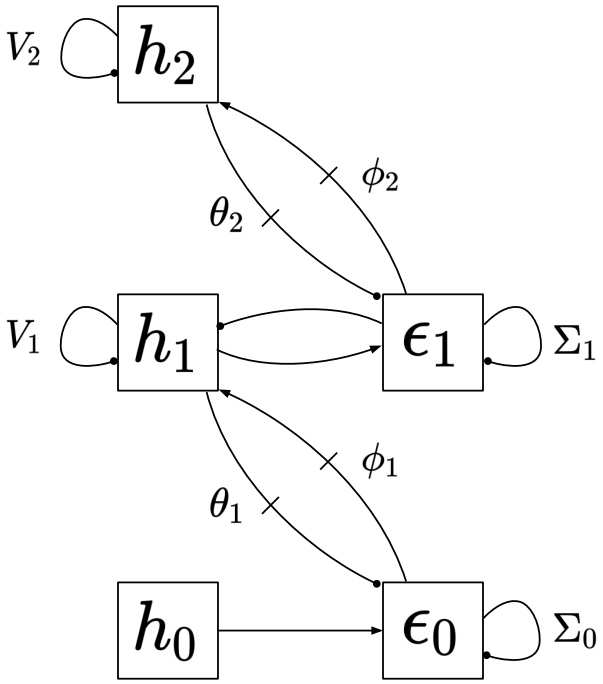
More recently, (Ororbia and Kifer, 2020) have proposed a PC model labeled Generative Neural Coding Network (GNCN). This model comprises additional lateral inhibition mechanisms parameterized by matrices V_i to entail sparse activations inside the representation populations. Their method reproduces the usual PC connection pattern between prediction error populations and representation populations. To improve the biological plausibility of their model, they suggest learning the feedback weights of the model according to the same learning rule used for the top-down weights, but transposed. This difference allows the feedback weights to be initialized randomly but still converge through learning towards the transpose of the forward weights. We can also note that they use a less constrained version of the PC algorithm we have derived from the FEP, in which the covariance matrices $\mathbf{\Sigma}_i$ of the generative model are not assumed to be diagonal. This entails a recurrent connection pattern in the prediction error neural populations, that we represented in figure 2.10b.

The following models do not per se implement PC since the propagation of prediction error is only used as a learning mechanism and no inference is performed. Still, we included them for comparison, as the described mechanisms for learning align to some extent to the learning method prescribed by PC.

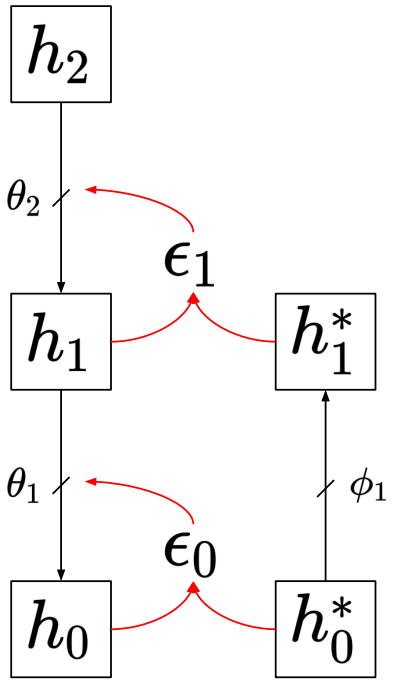
(Lee et al., 2015) propose an alternative to the BP learning algorithm where intermediary targets are defined at each layer. The model parameters at each layer are optimized following a local update rule aiming at minimizing the discrepancy between the layer prediction and the target defined for this layer. If relevant targets at each layer are provided to the model, this learning scheme approximates gradient descent on the output prediction error while not relying on the chain rule (i.e. BP). If the targets used at each layer were estimated using a PC inference process, the method described in this paper would be very close to PC learning. Though, they suggest estimating the targets at each layer using an auto-encoder architecture where the bottom-up computations are learned approximations of the inverses of top-down computations at each



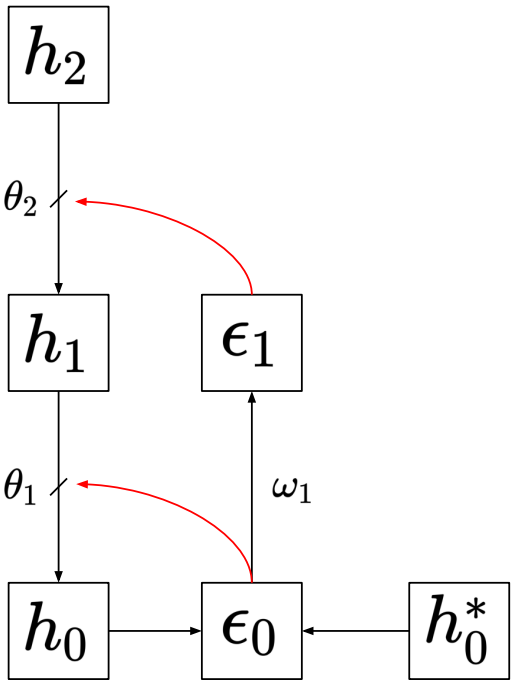
(a) PC network.



(b) Generative Neural Coding Network.



(c) Difference Target Propagation.



(d) Feedback Alignment.

Figure 2.10: Graphical representations of different neural networks architectures and learning methods related to PC. Black arrows designate synaptic connections, while red arrows denote learning computations. The mark on some of the synaptic connections indicate that the corresponding weights are learned. In all models, $\boldsymbol{\theta}_i$ denotes top-down weights, $\boldsymbol{\theta}_i^\intercal$ denotes the transpose of top-down weights used in the feedback pathway, $\boldsymbol{\phi}_i$ denotes learned feedback weights, and ω_i denotes random feedback weights.

layer. This hierarchical auto-encoder-like architecture is represented in figure 2.10c.

Investigating alternatives for BP, (Lillicrap et al., 2016) have shown that random feedback weights could propagate error signals in neural networks. According to this method, labeled feedback alignment, populations of neurons encode errors signals at each layer, as in PC. However, they show that the feedback connections do not have to be set according to the forward connections (and thus avoid the weight transport problem), and do not have to be learned through other mechanisms. Geometrically, random feedback connections can transport error information if they are to some extent aligned with the transposed forward weights. In their experiment, this soft alignment is observed as a side effect of the learning of the forward connections. The proposed algorithm performs similarly to BP in the presented experiments. We have represented the feedforward neural network as well as the separate error pathway in figure 2.10d.

Extending this idea is the method presented in (Nøkland, 2016). In this work, on top of providing explanations on the mechanisms at stake in feedback alignment, the authors generalize it to different feedback connection schemes. Especially, the direct feedback alignment algorithm suggests directly communicating the output error to any hidden layers through random feedback connections, without circulating through all the intermediate layers.

The PC architecture typically extends a multi-layer perceptron with additional prediction error neurons and modified connection patterns. This transformation can in fact be applied to other types of feedforward architectures, as thoroughly performed in (Millidge et al., 2020a). One of the early works implementing PC on typical deep learning architecture is the PredNet model presented in (Lotter et al., 2016). This work proposes a deep learning architecture where representations are built based on the propagation of prediction error. The described model does not strictly implement PC, as they compute the prediction error using a variable that differs from the current layer latent state variable. Both the top-down and bottom-up weights of the architecture are learned using BPTT. The propagation of prediction error as part of the neural dynamics only serves as an inference process, from which learning is decorrelated. The generative model is composed of CNNs combined with LSTMs to account for the temporal dynamics of the observed signal to model.

In this chapter, we have shown how PC transforms a feedforward generative model into an RNN that can perform inference and learning using online and local rules. It is important to note the difference between the recurrence involved in PC networks and the recurrence involved in neural network architectures dealing with temporal signals. The combination of the dynamic inference process, and the temporal processing imposed by the sequential nature of the processed data, is at the center of the next chapter.

We conclude this section about the algorithms related to PC by a table highlighting how those models implement some of the properties attributed to PC (table 2.1).

Model	Error neurons	Feedback weights	Precision weighting	Inference
PC (Rao and Ballard, 1999)	✓	Transposed	✓	✓
PC (Friston and Kiebel, 2009)	✓	Transposed	✓	✓
GNCN (Ororbia and Kifer, 2020)	✓	Learned	✓	✓
PC-DIM (Spratling, 2008)	✓ (divisive)	Transposed	✗	✓
Target propagation (Lee et al., 2015)	✗	Learned	✗	✗
Feedback alignment (Lillicrap et al., 2016)	✓	Random	✗	✗
Direct feedback alignment (Nøkland, 2016)	✓	Random	✗	✗

Table 2.1: Variants of the PC architecture.

Chapter 3

Sequence memory modeling

3.1 Introduction

In this chapter, we use the concepts of RNNs and PC introduced previously in order to formulate a candidate solution to the problem of sequence memory learning.

Here is an outline of this chapter:

- First, section 3.2 proposes an overview of the existing ANN based methods for sequence memory modeling.
- Then, section 3.3 presents several implementations of RNNs that we derive from the FEP.
- In section 3.4 we experiment with these models.
- Finally, section 4.4 discusses the obtained results and concludes this chapter.

3.2 Related work

We have previously explained that sequential data have two characteristic dimensions: the sequence length, and the dimension of each item in the sequence. Models that take into account this structure in the data as an inductive bias, should perform better than those not using this information.

For this reason, even though theoretically, any function approximation method could be used as a sequence memory model, we focus on approaches that take the temporal nature of sequential data into account.

Additionally, this thesis takes inspiration from the PC theory and consequently focuses on neural network models. In this section, we review existing approaches for sequence memory modeling that are based on ANNs.

In section 3.2.1, we describe some feedforward neural networks architectures that can be used for sequence memory modeling. Section 3.2.2 presents some advanced RNN models that have been proposed in the literature to tackle the problems of exploding and vanishing gradients. In section 3.2.3, we introduce the RC approach that addresses these issues differently. Finally, in section 3.2.4, we review existing RNN models taking inspiration from the PC theory.

3.2.1 Feedforward architectures

This section presents different feedforward neural network architectures that can be used to learn sequence memories. This part is intentionally brief, the thesis work focusing instead on recurrent architectures, that we study in depth in sections 3.2.2 and 3.2.3.

3.2.1.1 Perceptron and tabular case

The simplest neural network model that we can discuss is the one-layer perceptron. In this model, the output variable is a linear mapping of the input variable.

In the general case, our data set for sequence learning is composed of pairs $\big(\boldsymbol{c}, (\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)\big)$ where $\boldsymbol{c}$ is an identifier of the trajectory $(\boldsymbol{x}_1, \dots, \boldsymbol{x}_T)$. If no embedding for $\boldsymbol{c}$ is provided (this is the

case in most of the work we present), $\boldsymbol{c}$ is considered to be the one-hot embedding of dimension p with index k, where p denotes the number of trajectories in the data set, and k denotes the index of the trajectory in the data set. This means that $\boldsymbol{c}$ is a null vector of dimension p, except for the coefficient k that takes the value 1.

We can use the perceptron as a model of our sequence memory. The input of the network is the one-hot embedding of the index k of the trajectory to learn, and the desired output is the trajectory flattened into a vector of dimension $T \times d$. Consequently, the weight matrix of the perceptron is of dimension $(T \times d, p)$. Given a one-hot encoded input $\boldsymbol{c}$, the network outputs the k-th column of the weight matrix. We can see that this model amounts to learning a tabular memory of the sequences. The weight matrix is equivalent to a table containing in each column k the value of the trajectory.

As we can see, this model is too simple to be of any use. Adding hidden layers should improve its generalization capability, but this still would not be a very satisfactory solution, as it does not exploit in any way the temporal structure of the data to generate.

3.2.1.2 Convolutional neural networks

As previously explained, a sequence memory model should have an output of dimension (T, d) where T is the sequence length, and d is the dimension of each item in the sequence. Consequently, we are looking for models that can exploit this known structure.

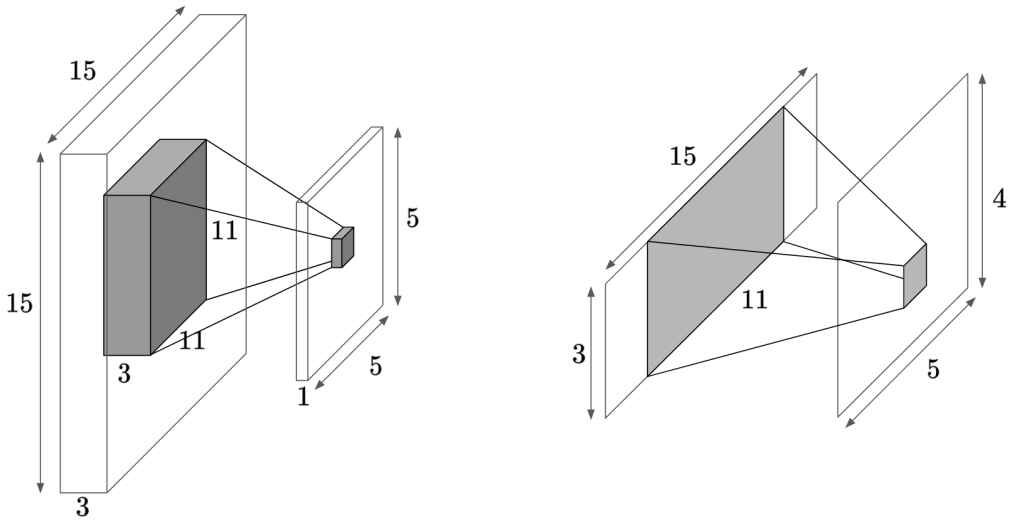


Figure 3.1: 2D convolution and 1D convolution.

Convolutional Neural Networks (CNNs) form a family of feedforward neural networks in which the connection pattern between two layers exploits the structure of the encoded variables. They have been used mostly in the case of image processing (Krizhevsky et al., 2012; He et al., 2016), where variables have a 3D structure, with one horizontal dimension, one vertical dimension, and one dimension for the color channels (or more generally a feature dimension).

The exploitation of this structure is represented in figure 3.1. The first figure represents a convolutional layer performing a 2D convolution on an input of dimension $(15, 15, 3)$. With each neuron in the output layer is associated a volume of dimension $(11, 11, 3)$ from the input layer, that we can call receptive field. The neuron's activation is computed as the dot product between the flattened volume and a synaptic weight vector (the model parameters), on which we apply an activation function. This weight vector is shared across all the neurons of the output layer, the differences in the output activations thus only coming from the differences in their receptive fields.

This type of models is interesting for our problem since it exploits the known structure of the available data. In the right figure, we have a convolutional layer performing a 1D convolution on an input layer of dimension $(15, 3)$. 1D convolutions of this type have been used for sequence processing, for instance in natural language processing (Collobert and Weston, 2008; Kalchbrenner et al., 2014). The dimension on which the receptive field (represented horizontally) is moving is the

temporal dimension, while the other dimension (represented vertically) is the feature dimension. This convolutional layer transforms the input sequence into a sequence of dimension $(5, 4)$.

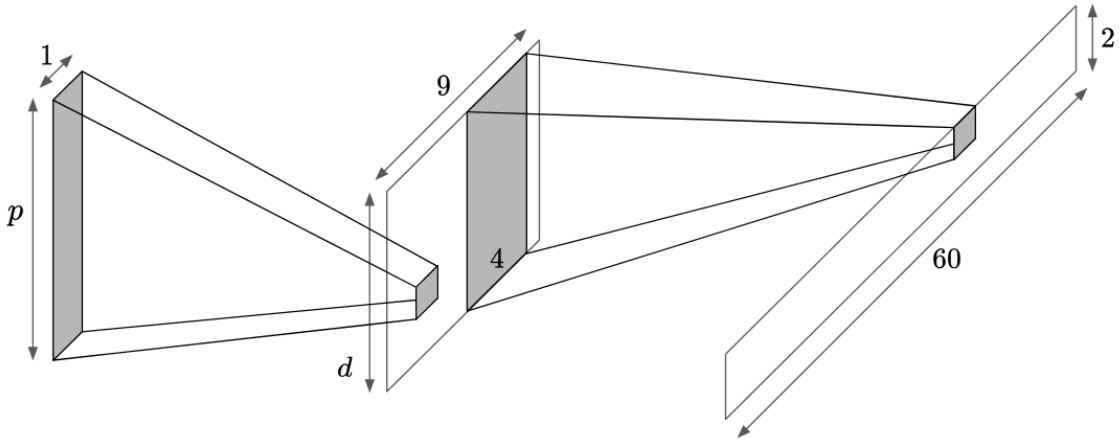


Figure 3.2: Deconvolutional neural network.

By playing with parameters such as *padding*, *stride*, and the number of output features, it is possible to design 1D convolutional layers where the output length is greater that the input length. These types of layers have been labeled *deconvolutional* or *transposed convolutional* layers. By stacking several deconvolutional layers, we can obtain a neural architecture associating a sequential output of dimension $(60, 2)$ with an input vector of dimension p. Such an architecture is represented in figure 3.2. We could train a sequence memory model using this architecture.

3.2.1.3 Attention mechanisms

Attention mechanisms have gained a lot of interest in recent years, in particular for sequence-to-sequence tasks. These mechanisms are used to model the dependency between input and output items without regard to their position in the sequences. In (Bahdanau et al., 2015), the authors propose an RNN using an attention mechanism to align the current output prediction with preferred parts of the input sequence.

The Transformer architecture (Vaswani et al., 2017) was introduced as an improvement on RNN based sequence-to-sequence tasks. This model has since been very successful at machine translation and language modeling (Devlin et al., 2018; Brown et al., 2020). The Transformer architecture takes into account the temporal structure of the data without caring about the chronology of the inputs or the outputs. Both sequences are considered as sets of items that can be queried thanks to the attention mechanisms, without regard to their position in the sequence. Since this information can still be useful, it is suggested to augment the items representations using a technique called positional encoding.

In the Transformer architecture, the decoder uses two attention mechanisms. First, it uses self-attention to generate a representation based on the past values of the output sequence. The second attention mechanism uses this representation as queries to attend the encoded inputs.

For the question of sequence generation, we cannot directly use these methods because the input $\boldsymbol{c}$ of the model is not sequential. Still, we can modify the encoder in order for it to generate a set of vectors that can be queried by the second attention mechanism of the decoder. A simple implementation of a sequence memory model using attention mechanisms is represented in figure 3.3. Note that this model only serves as an example, and is not supposed to represent the state of the art in using attention mechanisms for sequence modeling.

An important drawback of these models is that they do not scale very well with increased trajectory lengths. In RNNs, the computation cost of predicting one value $\hat{\boldsymbol{x}}_{t+1}$ is constant. Here, since the model has to perform attention on the complete past trajectory, the computation cost for the prediction of $\hat{\boldsymbol{x}}_{t+1}$ is proportional to t.

Still, the field of machine learning and particularly natural language processing has seen impressive advances thanks to these techniques. However, since this is not the focus of this thesis, we now turn away from CNNs and attention mechanisms and instead present a more in depth review

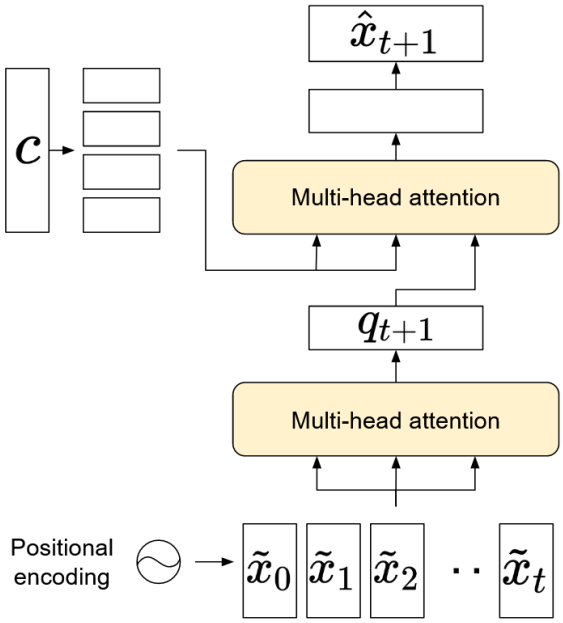


Figure 3.3: Example of ANN model for sequence modeling using attention mechanisms.

of RNN architectures.

3.2.2 Advanced RNN models

As we have seen in chapter 2, an important drawback for the learning of RNN parameters is the length of the unrolled computational graph. Propagating information from an input at time t to an output at time $t + T$ traverses a long path in the computational graph. Learning long-range dependencies can thus be difficult, and some approaches for RNN modeling have tried to address this issue by using gating mechanisms.

In this line of research, Long Short-Term Memories (LSTMs) (Hochreiter and Schmidhuber, 1997) propose to structure the RNN state into a hidden state $\boldsymbol{h}_t$ and a cell state $\boldsymbol{c}_t$. The cell state allows some information to flow in time unchanged, thanks to gating mechanisms. The model is described by the following set of equations (we recall that bias parameters are omitted for simplicity):

$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_{ih} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{iy} \cdot \boldsymbol{y}_t) \tag{3.1}$$
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_{fh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{fy} \cdot \boldsymbol{y}_t) \tag{3.2}$$
$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_{oh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{oy} \cdot \boldsymbol{y}_t) \tag{3.3}$$
$$\hat{\boldsymbol{c}}_t = \tanh(\boldsymbol{W}_{ch} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{cy} \cdot \boldsymbol{y}_t) \tag{3.4}$$
$$\boldsymbol{c}_t = \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \odot \hat{\boldsymbol{c}}_t \tag{3.5}$$
$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t) \tag{3.6}$$

The input and forget gates, controlled by the variables $\boldsymbol{i}_t$ and $\boldsymbol{f}_t$, supervise the update of $\boldsymbol{c}_t$. The output gate, controlled by the variable $\boldsymbol{o}_t$, filters the information from $\boldsymbol{c}_t$ that is available for the next time step computations. Other works such as (Greff et al., 2015) have studied the relative impact of the different gating mechanisms of the LSTM models. Some simpler RNN models with gating mechanisms have been proposed in the literature, such as the Gated Recurrent Unit (GRU) (Cho et al., 2014).

The GRU model uses two gating operations: one gate filters the information from the past hidden that can be used to predict the future hidden state, while the other controls how much the hidden state should be updated based on this prediction. The GRU model is described by the following equations:

$$\boldsymbol{r}_t = \sigma(\boldsymbol{W}_{rh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{ry} \cdot \boldsymbol{y}_t) \tag{3.7}$$
$$\boldsymbol{u}_t = \sigma(\boldsymbol{W}_{uh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{uy} \cdot \boldsymbol{y}_t) \tag{3.8}$$
$$\hat{\boldsymbol{h}}_t = \tanh\big(\boldsymbol{W}_{hh} \cdot (\boldsymbol{r}_t \odot \boldsymbol{h}_{t-1}) + \boldsymbol{W}_{hy} \cdot \boldsymbol{y}_t\big) \tag{3.9}$$
$$\boldsymbol{h}_t = \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \odot \hat{\boldsymbol{h}}_t \tag{3.10}$$

Finally, considering that the update gate controlled by u_t seems to be the most important (Greff et al., 2015), the simplest RNN model with gating that we consider is the Update Gate RNN (UGRNN) (Collins et al., 2017). The UGRNN model is described by the following equations:

$$\boldsymbol{u}_t = \sigma(\boldsymbol{W}_{uh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{uy} \cdot \boldsymbol{y}_t) \tag{3.11}$$
$$\hat{\boldsymbol{h}}_t = \tanh(\boldsymbol{W}_{hh} \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{hy} \cdot \boldsymbol{y}_t) \tag{3.12}$$
$$\boldsymbol{h}_t = \boldsymbol{u}_t \odot \boldsymbol{h}_{t-1} + (1 - \boldsymbol{u}_t) \odot \hat{\boldsymbol{h}}_t \tag{3.13}$$

An orthogonal way of improving RNN performances is to stack several RNN layers. In this method, the output of the layer l is used as input for the layer $l-1$ at each time step. This method is widely used and allows different layers to capture different abstraction levels or time scales of the target sequences.

An example of this method is the Multiple Timescales RNN (MTRNN) model (Yamashita and Tani, 2008) that stacks several TRNN layers with different time constants $\tau^{(l)}$. Layers that are higher in the hierarchy are associated with higher time constants, and thus exhibit slower dynamics. The idea behind this model is that more abstract or general features of the signal might evolve at a slower pace. We can show that the direct dependency between $\boldsymbol{h}_{t+T}^{(l)}$ and $\boldsymbol{h}_t^{(l)}$ decreases exponentially in $(1 - \frac{1}{\tau^{(l)}})^T$. Consequently, higher values of $\tau^{(l)}$ in the upper layers mean a better backpropagation through time of gradients in these layers. The following equations describe an MTRNN model composed of two layers $\boldsymbol{h}^f$ and $\boldsymbol{h}^s$ (for fast and slow), associated respectively with the time constants τ_f and τ_s:

$$\boldsymbol{h}_t^s = (1 - \frac{1}{\tau_s})\boldsymbol{h}_{t-1}^s + \frac{1}{\tau_s}\big(\boldsymbol{W}_{ss} \cdot \tanh(\boldsymbol{h}_{t-1}^s) + \boldsymbol{W}_{sf} \cdot \tanh(\boldsymbol{h}_{t-1}^f)\big) \tag{3.14}$$
$$\boldsymbol{h}_t^f = (1 - \frac{1}{\tau_f})\boldsymbol{h}_{t-1}^f + \frac{1}{\tau_s}\big(\boldsymbol{W}_{fs} \cdot \tanh(\boldsymbol{h}_{t-1}^s) + \boldsymbol{W}_{ff} \cdot \tanh(\boldsymbol{h}_{t-1}^f) + \boldsymbol{W}_{fy} \cdot \boldsymbol{y}_t\big) \tag{3.15}$$
$$\tag{3.16}$$

Closely resembling the MTRNN is the Clockwork RNN (CWRNN) model introduced in (Koutnik et al., 2014). The authors suggest structuring the hidden state of the RNN into different modules processing inputs at different time-scales. Thanks to this factorization, the model can decompose the signal into components with different paces. The main difference with the MTRNN model lies in the fact that this architecture is not hierarchical, all modules are connected with the input and output layers. The Structurally Constrained RNN (SCRNN) model proposed in (Mikolov et al., 2014) can be seen as a special case of CWRNN using two modules with one of the recurrence matrices being equal to the identity.

Stacking a large number of layers increases the length of the path between variables in the upper layers and variables in the lower layers in the computational graph. This can lead to another instance of vanishing gradient, where gradient flows vanish while propagating through the hierarchy instead of time. Based on the success of gated recurrent architectures, depth gating (Srivastava et al., 2015) has been proposed to address this issue. In (Yao et al., 2015), the authors implement a depth gate on a stack of LSTMs. In (Collins et al., 2017), one of the models put forward by the authors combines the use of depth and update gates (as in the UGRNN and GRU models), and is labeled Intersection RNN (+RNN).

We have seen how to improve RNNs by building a hierarchy or using gating mechanisms. Here we discuss another promising research direction for RNN models. Studying RNNs from a dynamical systems point of view led us to better understand the exploding and vanishing gradient problems, but this can also help us designing models mitigating these. In this line of research, some approaches

have proposed to constrain the recurrent matrix of RNN models in order to limit the vanishing or explosion of gradients during BP. Specifically, the evolution of the gradient amplitudes during BP depends highly on the Jacobian $J^{ij} = \frac{\partial h^i_{t+1}}{\partial h^j_t}$. Constraining this matrix to have unitary eigenvalue helps preventing vanishing or exploding gradients. In particular, it has been proposed to initialize the recurrent weights using the identity matrix (Le et al., 2015) or orthogonal matrices (Mishkin and Matas, 2016). (Vorontsov et al., 2017) have investigated models constraining or encouraging recurrent weight matrices to remain orthogonal, and have shown that it can have detrimental effects on trainability. Finally, (Chang et al., 2019) proposed the Antisymmetric RNN, combining the idea of skip connections in RNNs (Yue et al., 2018) and antisymmetric weights, that has shown competitive performances with gated models. The Antisymmetric RNN model can be described by the following equation:

$$\boldsymbol{h}_t = \boldsymbol{h}_{t-1} + \epsilon \tanh\left((\boldsymbol{W}_{hh} - \boldsymbol{W}_{hh}^\intercal - \gamma \mathbb{I}) \cdot \boldsymbol{h}_{t-1} + \boldsymbol{W}_{hy} \cdot \boldsymbol{y}_t\right) \tag{3.17}$$

where $\boldsymbol{W}_{hh} - \boldsymbol{W}_{hh}^\intercal$ is antisymmetric by construction, and comprises only $d_h \times (d_h - 1)/2$ parameters to be trained.

We can make several comments regarding all the RNN models that we have presented here. First, the described models do not generate an output. To adapt these model for sequence generation, we need to add an output layer computing a prediction $\boldsymbol{x}_t$ from the network hidden state $\boldsymbol{h}_t$:

$$\boldsymbol{x}_t = \boldsymbol{W}_o \cdot \boldsymbol{h}_t \tag{3.18}$$

Note that in some models $\boldsymbol{h}_t$ corresponds to he pre-activation value of the hidden state and that we instead use $\tanh(\boldsymbol{h}_t)$ in the last equation.

Second, the models' computations are based on a temporal input $\boldsymbol{y}_t$. In our sequence memory modeling task, no such input is provided. To use these models, we can either consider this input to be null, or reinject the past prediction $\boldsymbol{x}_{t-1}$ as input for the next step:

$$\boldsymbol{y}_t = \boldsymbol{x}_{t-1} \tag{3.19}$$

This second method is often used for sequence generation tasks, and additionally allows using the target values of $\boldsymbol{x}_t$ as input during training, a method that has been labeled as teacher forcing.

3.2.3 Reservoir computing

We have described many RNN models that were designed in order to improve the difficult temporal assignment problem. Being able to learn the delayed dependency between inputs and outputs to an RNN is essential for a lot of tasks. However, for the question of sequence generation, it has been argued that if the recurrent weights are initialized in a way that ensures complex hidden state dynamics, then training the output layer might be enough to properly approximate the target temporal patterns.

Completely avoiding the problem of learning recurrent weights, a family of approaches has emerged in parallel from the field of computational neurosciences in the form of Liquid State Machines (Maass et al., 2002), and from the field of machine learning in the form of Echo State Networks (ESN) (Jaeger, 2001). These models, later brought together under the label of Reservoir Computing (Verstraeten et al., 2007; Lukoševičius and Jaeger, 2009), discard the difficulties of learning recurrent weights by instead developing techniques to find relevant initializations of these parameters.

Typically, the recurrent connections are set in order for the RNN to exhibit rich non-linear (and sometimes self-sustained) dynamics, that are decoded by a learned readout layer. If the dynamics of the RNN activation are complex enough (e.g. they do not converge too rapidly towards a point attractor or limit cycle attractor), various output sequences can be decoded from those. Training RC models then comes down to learning the weights of the readout layer, which is an easier optimization problem that can be tackled with several algorithms. This output layer can for instance be trained using stochastic gradient descent, without the need for BP. The FORCE algorithm (Sussillo and Abbott, 2009) improves this learning by running an iterative estimate of the correlation matrix of the reservoir activations.

Another interesting learning mechanism is presented in (Jaeger, 2014a; Jaeger, 2014b) under the name of Conceptors. This method exploits the fact that the reservoir state (or hidden state)

dynamics triggered by an input pattern is typically bounded to a certain subspace of lower dimension. By identifying the subspace for each possible input pattern, it is possible to decorrelate the training of each target trajectory by focusing learning on the readout connections that come from the corresponding reservoir state subspace (called Conceptor). This method allows training a sequence memory where the learning of a new pattern has limited interference with already learned ones.

Mathematically, this method can be implemented using only online computations. The Conceptor $\boldsymbol{C}$ associated with some input can be defined as the matrix corresponding to a soft projection on the subspace where the reservoir dynamics lie when stimulated with this input. The softness of this projection is controlled by a positive parameter α called aperture. Low values of α induce hard projections onto the subspaces, while high values of α induce Conceptor matrices close to the identity matrix, thus not performing any projection. This matrix $\boldsymbol{C}$ can be computed using the reservoir state correlation matrix $\boldsymbol{R}$ estimated online based on the reservoir dynamics:

$$\boldsymbol{C} = \boldsymbol{R} \cdot (\boldsymbol{R} + \alpha^{-2}\mathbb{I})^{-1} \tag{3.20}$$

$$\boldsymbol{R}_{t+1} = \big(1 - \frac{1}{t+1}\big)\boldsymbol{R}_t + \frac{1}{t+1}\big(\boldsymbol{h}_t \cdot \boldsymbol{h}_t^\intercal\big) \tag{3.21}$$

In a continual learning setting, for each new task, we can compute the Conceptor corresponding to the complement of the subspace in which lie the previously seen reservoir states, as $\mathbb{I} - \boldsymbol{C}$. This Conceptor is used to project the new reservoir states into a subspace orthogonal to the subspace in which lie the previously seen reservoir states. Learning is then performed only on the synaptic weights involving the components of this subspace. We can finally also note that the use of Conceptors is not limited to RC, as it has been for instance used jointly with BP in (He and Jaeger, 2018).

Because RC methods do not rely on BP of the output error, there is no neural mechanism transporting information from the target signal to the input weights of the RNNs. To perform learning of the input weights of reservoirs, (Pitti et al., 2017) propose to use a secondary learning system implementing random search on the reservoir inputs. Experimentally, this random search can converge quickly towards a local optimum but can also interfere negatively with the learning of the readout weights.

3.2.4 Applying PC to RNN models

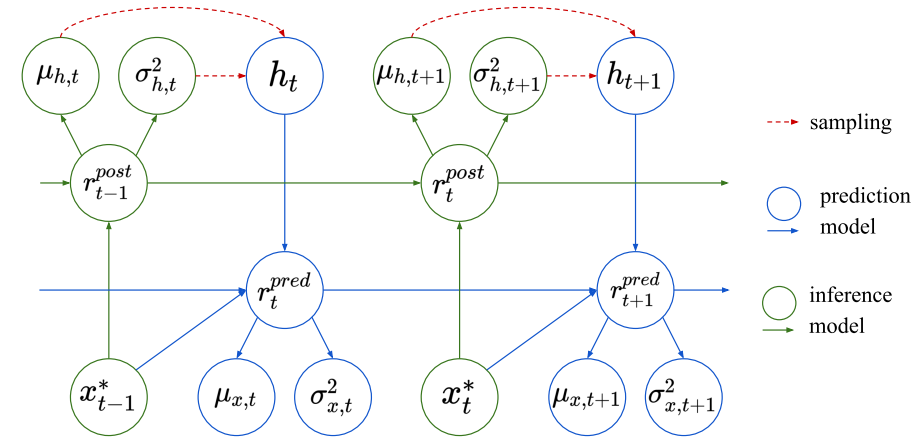
In the previous chapter, we have introduced the PC theory and presented the associated neural network architectures, inference mechanisms, and learning rules.

We have seen how PC casts a feedforward generative model into an RNN with temporal dynamics induced by the iterative inference process. Applying this transformation to temporal generative models such as the sRNN model presented above introduces a new complexity as there is an interference between the dynamics of the generative model and the dynamics of the inference mechanism.

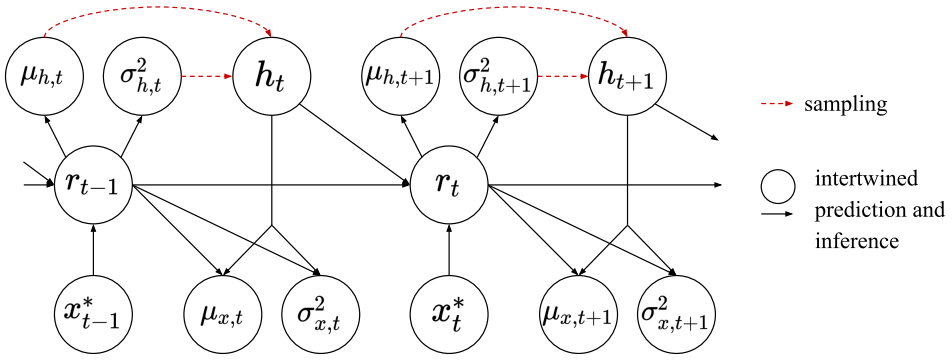
The different methods reviewed in this section are summarized in table 3.1.

In (Millidge et al., 2020a), the authors apply the PC transformation on temporal generative models such as LSTMs, without considering this issue. All the variables in the generative models are associated with a prediction error variable and feedback connections are added in parallel to the whole LSTM computational graph. However, if we consider the case of a simulated agent perceiving sensory observations in real-time, inferring past latent variables $\boldsymbol{H}_{t-k}$ based on current sensory observation $\boldsymbol{x}_t$ would require storing the past hidden state of neural networks. On the other end, the inference process is not a one-shot mechanism. The iterative update of latent variables supposes consecutive replays of the whole sequence of sensory observations $\{\boldsymbol{x}_0, \cdots, \boldsymbol{x}_T\}$ until convergence. In other words, such an application of PC on temporal generative models considers a feedforward version of the generative model obtained by unrolling the temporal computations, and thus decorrelates the *inference time* with the *temporal generative model time*. In the case of an embodied agent, such an assumption might not be justified, as we expect the inference process to happen online, continuously, while new perceptual information is acquired by the agent sensors. In this section, we review RNN models related to the PC theory and not relying on this assumption.

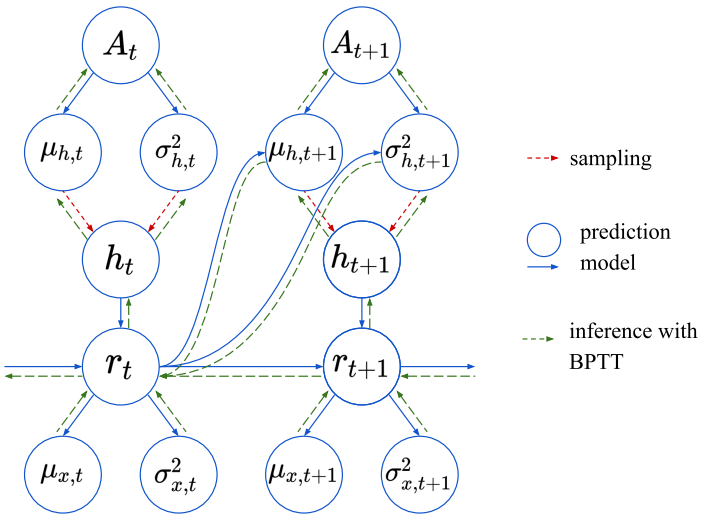
(Bayer and Osendorfer, 2015) propose to extend the variational Bayes auto-encoder model from (Kingma and Welling, 2014; Rezende et al., 2014) to temporal generative models. The generative



(a) STORN model.



(b) VRNN model.



(c) PVRNN model.

Figure 3.4: Models integrating variational inference as performed in VAEs to the design of RNNs. The inference model is represented in green, while the prediction model is represented in blue. In the second model, prediction and inference are intertwined and the synaptic connections are thus represented in black. Red dashed arrows represent the sampling of $\boldsymbol{h}$ according to the multivariate Gaussian of parameters $\boldsymbol{\mu}_h$ and σ^2_h. BP through this sampling operation is possible thanks to the reparameterization trick. Note that the notations we use here differ from the ones in the original articles.

model $p(\boldsymbol{x}|\boldsymbol{h};\boldsymbol{\theta})$, as well as the recognition density model $q(\boldsymbol{h};\phi)$ are each implemented by an sRNN. The recognition model performs a one-shot inference of the temporal latent random variable $\boldsymbol{H}_t$ based on a recognition RNN hidden state and the previously observed variable $\boldsymbol{x}_{t-1}$. On the other hand, BPTT is used to perform learning. This model, labeled Stochastic RNN (STORN), is very close to the Variational Recurrent Auto-Encoder model proposed in (Fabius and Amersfoort, 2015) and the DRAW model proposed in (Gregor et al., 2015). The differences between these three models only lie in the structure of the generative model $p(\boldsymbol{x}_t|\boldsymbol{h}_t)$. The inference and learning methods described are exactly the same. The neural architecture described in (Bayer and Osendorfer, 2015) is represented in figure 3.4a.

Another related model is the Variational RNN (VRNN) model proposed in (Chung et al., 2015). In this method, the prior $p(\boldsymbol{h}_t)$ of the latent variable is made dependent on the hidden state of an RNN. The model departs from the traditional VAE as usually the prior distribution on the latent variance is only used for learning, not during the inference process. Additionally, contrary to the STORN model, the VRNN does not rely on a secondary RNN. They propose several versions of this model, using either a multivariate Gaussian or a Gaussian mixture model for $p(\boldsymbol{x}_t|\boldsymbol{h}_t)$. The described neural architecture is represented in figure 3.4b. Note that on this figure we do not represent the approximate posterior distribution parameters that are computed based on the hidden state r_t and the observation $\boldsymbol{x}_t^*$. Both these models (STORN and VRNN) are learned using the same loss function combining an output prediction error and Kullback-Leibler divergence as a measure of the distance between the prior and approximate posterior on the latent variable $\boldsymbol{H}$.

Another model drawing a connection between VAEs and RNNs is described in (Gemici et al., 2017). In this work, the authors present different methods to integrate VAEs into RNNs. In one of the possible models, at each time step, an RNN prescribes a prior distribution $p(\boldsymbol{h}_t)$ for the latent variable, that is used for prediction. After observation of $\boldsymbol{x}_t$, the latent distribution is updated based on a recognition density. This approximate posterior latent distribution is then used as input for the recurrence computation of the RNN. Additionally, they add an external addressable memory to the whole architecture, that we do not describe here.

A drawback of these models that has been pointed out is that the recognition density (or approximate posterior density) $q(\boldsymbol{h}_t)$ only depends on past and current observed variables $\boldsymbol{x}_{\leq t}$. The inference mechanism cannot account for events that are observed later to adjust its belief of the past latent variables. To deal with this issue, several models (Fraccaro et al., 2016; Shabanian et al., 2017; Goyal et al., 2017) have proposed to use a bidirectional RNN architecture where the recognition density parameters at each time step depend on the recurrent step of a backward RNN. (Girin et al., 2020) provide a recent and more complete review of models applying the VAE principle on sequential generative models. They label this class of models as Dynamical Variational Auto-Encoders (DVAE).

To optimize the parameters of the approximate posterior $q(\boldsymbol{h}_t)$ according to future observations $\boldsymbol{x}_{>t}$, (Ahmadi and Tani, 2019) propose to use the BPTT algorithm, which is more often used for learning. The distribution $q(\boldsymbol{h}_t)$ at each time step is made dependent on some parameter A_t that can be optimized for each target sequence by backpropagating the gradient of the prediction error through time. Contrary to the previously presented model that proposes to perform variational inference through the use of a forward or backward RNN, here inference is not part of neural dynamics but assimilated to the learning algorithm. In this article, the method is applied to an MTRNN model, for simplicity, we represented in figure 3.4c only one layer of the architecture described in (Ahmadi and Tani, 2019).

This inference method based on BPTT, also called Error Regression Scheme (ERS), had already been used to infer hidden states of an MTRNN in (Ahmadi and Tani, 2017), or the parametric bias variable of RNN models augmented with an additional inferable static variable (Ito and Tani, 2004). In ERS, we can define a time window W and a number of iterations I for the inference process through BPTT at each time step. If we consider that the latent variable is the hidden state of the RNN $\boldsymbol{h}_t$, then this method proposes to approximate the true posterior by regressing with BPTT, limited to I iterations on a time window of length W, the value of $\boldsymbol{h}_t$. In the case where $W = 1$, this algorithm can be performed using only information available at time step t, with no need for future observations.

So far, we have presented models proposing different implementations of variational Bayes onto temporal generative models, that take inspiration from the VAE architecture. All the models that consider a temporal backward mechanism to infer past latent variables, whether it is with backward RNNs, or with BPTT, can not perform inference *online*. To perform inference *online*, the inference

of the latent variable at time t has to only take into account currently available neural activations, such as the current observation $\boldsymbol{x}_t^*$. Online inference thus supposes an alignment between the clock of the temporal generative model and the clock of the inference process.

Some recurrent models inspired by PC actually implement online inference based on the output prediction error. The Parallel Temporal Neural Coding Network (P-TNCN) described in (Ororbia et al., 2020), implements PC in a fashion very similar to the GNCN model (Ororbia and Kifer, 2020) introduced in the previous chapter. The main difference is that the generative model is made temporal by adding recurrent connections onto the latent variable layer. The minimization of VFE also induces a learning rule for these additional recurrent weights, which are optimized in order to decrease the error between the prior hidden state $\boldsymbol{h}_t$ and its approximate posterior estimation $\boldsymbol{h}_t^*$. In P-TNCN, it is also suggested to modify the learning rule for the feedback weights responsible for the estimation of $\boldsymbol{h}_t^*$. The proposed rule optimizes these weights in order to minimize the increase of the prediction error on the hidden layer, instead of using the transpose of the learning rule used for the forward weights, as is suggested in (Rao and Ballard, 1999; Ororbia et al., 2020). In this model, and its version using the more classical learning rule for feedback weights, the inference and learning process can be performed online, using spatially and temporally local information.

Reinjecting the error feedback into the RNN hidden state, as is done in these models, has some drawbacks. This can make the RNN learning difficult as small corrections can have a large impact on the RNN dynamics if those are chaotic. Still, this error signal is necessary for learning the recurrent weights. Learning output weights while having a large feedback is problematic because the modification of the output weights causes delayed effects that are not taken into account in the learning rule. This kind of observation motivated the removal of feedback connections in the ESN model (Jaeger, 2001), and motivated the research for the FORCE algorithm that maintains a low prediction error throughout learning (Sussillo and Abbott, 2009). We have also verified experimentally that reducing the feedback amplitude improved learning in all RNN models implementing this feedback mechanism. Since this feedback is still necessary in some models to propagate the prediction error signal up into the hierarchy, we do not completely remove feedback connections during learning.

In the deep learning literature, some recurrent architectures such as the PredNet proposed in (Lotter et al., 2016) take inspiration from PC by building the bottom-up connection pathway upon prediction error signals rather than observations. However, in these models, learning is always performed using BPTT.

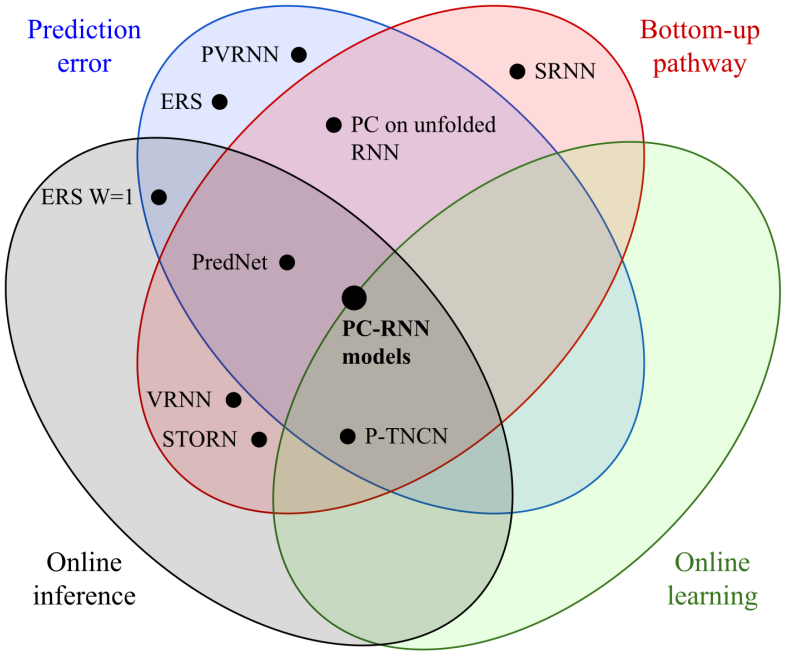


Figure 3.5: Venn diagram for the classification of RNN models performing variational inference. Each ellipse represents a classification criterion. The bidirectional arrow means that the PC-RNN models can belong to both categories depending on the learning algorithm.

Model	Bottom-up pathway	Online inference	Based on prediction error	Online learning
PC applied to unfolded RNN (Millidge et al., 2020a)	Transposed	✗	✓	✗
STORN (Bayer and Osendorfer, 2015)	Learned	✓	✗	✗
VRNN (Chung et al., 2015)	Learned	✓	✗	✗
SRNN (Fraccaro et al., 2016)	Learned	✗	✗	✗
PVRNN (Ahmadi and Tani, 2019)	✗	✗	✓	✗
ERS (Ahmadi and Tani, 2017)	✗	✗	✓	✗
ERS $W = 1$ (Ahmadi and Tani, 2017)	✗	✓	✓	✗
P-TNCN (Ororbia et al., 2020)	Learned	✓	✓	✓
PredNet (Lotter et al., 2016)	Learned	✓	✓	✗
PC-RNN with BPTT	Transposed	✓	✓	✗
PC-RNN with PC-based learning	Transposed	✓	✓	✓

Table 3.1: RNN models implementing a form of variational inference.

Finally, the FEP literature proposes to model temporal dynamics of latent variables using the principle of generalized coordinates. According to this idea, the latent variable $\boldsymbol{H}$ and the observed variable $\boldsymbol{X}$ are augmented by variables representing their higher-order of motions $\tilde{\boldsymbol{H}} = \{\boldsymbol{H}, \boldsymbol{H}', \boldsymbol{H}'', \cdots\}$ and $\tilde{\boldsymbol{X}} = \{\boldsymbol{X}, \boldsymbol{X}', \boldsymbol{X}'', \cdots\}$. In the next section, we design several RNN models derived from the free-energy formulation of PC, some of which implement this principle of generalized coordinates. To conclude this section, we summarize the discussed approaches in the table 3.1, listing all the presented RNN models implementing a form of variational Bayes inference. We classify the models according to four criteria. First, we note whether the models implement a bottom-up pathway as part of the neural circuit and whether the bottom-up weights are learned or transposed versions of the top-down weights. Second, we note whether they perform inference *online*. Third, we note whether the inference process is based on the prediction error signal. Finally, we note whether the learning method can also be performed online.

We can display this classification using a Venn diagram, as shown in figure 3.5. In this figure and in table 3.1, we have also inserted models labeled PC-RNN. PC-RNN denotes the family of models that are derived in the next section. As represented in this diagram, we position ourselves among models able to perform online inference, using a bottom-up pathway that propagates prediction

errors. The PC-RNNs are voluntarily placed on the border of the "Online learning" ellipse. Indeed, depending on the chosen learning algorithm, the models can either fall in one category or the other: learning can be performed using BPTT, which does not validate the online learning criterion, or using learning rules that we derive from the PC theory in the next section.

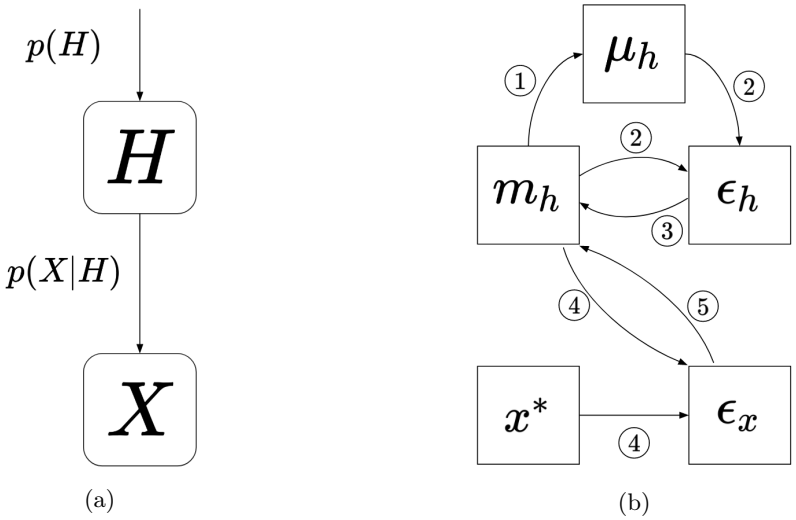
3.3 Methods

In order to build a sequence memory of temporal patterns, we want to design RNN models based on the PC theory. In this section, we design several models labeled with the same PC-RNN prefix, that are endowed with online inference and learning mechanisms directly minimizing VFE. We discuss several implementation choices, such as the use of generalized coordinates, or hidden causes layers, as well as some possible extensions of the proposed models in section 3.3.6.

We try to provide different representations of each derived model in order to have a clear understanding of their different aspects: a graphical representation of the probabilistic generative model, a neural network representation of the RNN, and the associated computational graph.

3.3.1 Simple recurrent model

In this section, we derive an RNN model from the equations of the FEP that we have previously introduced. This model is based on a simple probabilistic generative model, and a set of assumptions that we explain under way.



(a) (b)

Figure 3.6: Simple PC-RNN probabilistic model and neural network representation. **Left:** Graphical representation of the generative model. **Right:** Neural network representation of the resulting FE minimization process. The synaptic connections are numbered according to the order in which we perform the corresponding computations during a single time step.

The generative model is composed of two multivariate random variables $\boldsymbol{H}$, of dimension d_h, and $\boldsymbol{X}$, of dimension d_x. Typically, d_h has an order of magnitude of 50 and in most of our experiments, the output space is of dimension $d_x = 2$. The generative model is represented in figure 3.6a, and defined by the following equations:

$$p(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}_h, \sigma_h^2 \mathbb{I}_{d_h}) \tag{3.22}$$

$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{g}(\boldsymbol{h}), \sigma_x^2 \mathbb{I}_{d_x}) \tag{3.23}$$

The prior density $p(\boldsymbol{h})$ and the likelihood density $p(\boldsymbol{x}|\boldsymbol{h})$ are both assumed to be multivariate Gaussians. $\boldsymbol{\mu}_h$ and $\sigma_h^2 \mathbb{I}_{d_h}$ denote the prior density mean and covariance matrix, and $\boldsymbol{g}(\boldsymbol{h})$ and $\sigma_x^2 \mathbb{I}_{d_x}$ denote the likelihood density mean and covariance matrix. We assume that the covariance matrices are proportional to the identity matrices of dimension d_h and d_x.

The function $\boldsymbol{g}$ performs a linear mapping from the variable $\boldsymbol{H}$ on which we first apply the hyperbolic tangent function:

$$\boldsymbol{g}(\boldsymbol{h}) = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{h}) \tag{3.24}$$

where $\boldsymbol{W}_o$ is a matrix of dimension (d_x, d_h), corresponding to the output weights of our RNN model. To perform variational inference, we introduce a recognition density function $q(\boldsymbol{H})$:

$$q(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{m}_h, v_h \mathbb{I}_{d_h}) \tag{3.25}$$

This recognition density is also assumed to be Gaussian, with mean $\boldsymbol{m}_h$ and variance v_h. We have already shown in section 2.2.2 that using these assumptions, we could write the VFE as:

$$F(\boldsymbol{x}^*, \boldsymbol{m}_h) = E(\boldsymbol{x}^*, \boldsymbol{m}_h) + C \tag{3.26}$$

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h) = &\frac{1}{2\sigma_x^2}\|\boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)\|_2^2 + \frac{d_x}{2}\log(\sigma_x^2) \\ &+ \frac{1}{2\sigma_h^2}\|\boldsymbol{m}_h - \boldsymbol{\mu}_h\|_2^2 + \frac{d_h}{2}\log(\sigma_h^2) \\ &+ C \end{aligned} \tag{3.27}$$

We can now derive from this expression the gradient of the VFE with regard to the recognition density parameter $\boldsymbol{m}_h$. We denote by $\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h)}{\partial \boldsymbol{m}_{h,i}}$ the partial derivative with regard to the coefficient i of $\boldsymbol{m}_h$.

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h)}{\partial m_{h,i}} = &-\frac{1}{\sigma_x^2}\sum_{j=1}^{d_x}\big(x_j^* - g_j(\boldsymbol{m}_h)\big)\frac{\partial g_j(\boldsymbol{m}_h)}{\partial m_{h,i}} \\ &+ \frac{1}{\sigma_h^2}\big(m_{h,i} - \mu_{h,i}\big) \end{aligned} \tag{3.28}$$

By injecting the definition of $\boldsymbol{g}$ into this equation, we obtain:

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h)}{\partial m_{h,i}} = &-\frac{1}{\sigma_x^2}\sum_{j=1}^{d_x}\big(x_j^* - g_j(\boldsymbol{m}_h)\big)\big(1 - \tanh^2(m_{h,i})\big)W_o^{ij} \\ &+ \frac{1}{\sigma_h^2}\big(m_{h,i} - \mu_{h,i}\big) \end{aligned} \tag{3.29}$$

Which results in the following gradient descent update rule for $\boldsymbol{m}_h$:

$$\begin{aligned} \boldsymbol{m}_h \leftarrow \boldsymbol{m}_h &\underbrace{+\frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_h)\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_x\big)}_{\text{Bottom-up}} \\ &\underbrace{-\frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_h}_{\text{Top-down}} \end{aligned} \tag{3.30}$$

In this equation, we have introduced the notations $\boldsymbol{\epsilon}_x = \boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)$ and $\boldsymbol{\epsilon}_h = \boldsymbol{m}_h - \boldsymbol{\mu}_h$ that correspond to the prediction errors at different layers of the generative model. Finally, to make this model able to generate temporal patterns, we make the prior density mean $\boldsymbol{\mu}_h$ dependent on the value of the recognition density mean $\boldsymbol{m}_h$:

$$\boldsymbol{\mu}_h = (1 - \frac{1}{\tau})\boldsymbol{m}_h + \frac{1}{\tau}\boldsymbol{W}_r \cdot \tanh\big(\boldsymbol{m}_h\big) \tag{3.31}$$

where τ and $\boldsymbol{W}_r$ respectively correspond to the time constant and the recurrent weights of our RNN. Using equations 3.30 and 3.31, we can construct a system of equations guiding the temporal evolutions of the variables of our neural network:

$$\boldsymbol{\mu}_{h,t} = (1 - \frac{1}{\tau})\boldsymbol{m}_{h,t-1}^{post} + \frac{1}{\tau}\boldsymbol{W}_r \cdot \tanh\left(\boldsymbol{m}_{h,t-1}^{post}\right) \tag{3.32}$$

$$\boldsymbol{\epsilon}_{h,t} = \boldsymbol{m}_{h,t-1}^{post} - \boldsymbol{\mu}_{h,t} \tag{3.33}$$

$$\boldsymbol{m}_{h,t}^{prior} = \boldsymbol{m}_{h,t-1}^{post} - \frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_{h,t} \tag{3.34}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{W}_o \cdot \tanh\left(\boldsymbol{m}_{h,t}^{prior}\right) \tag{3.35}$$

$$\boldsymbol{\epsilon}_{x,t} = \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t \tag{3.36}$$

$$\boldsymbol{m}_{h,t}^{post} = \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\left(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\right) \odot \left(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\right) \tag{3.37}$$

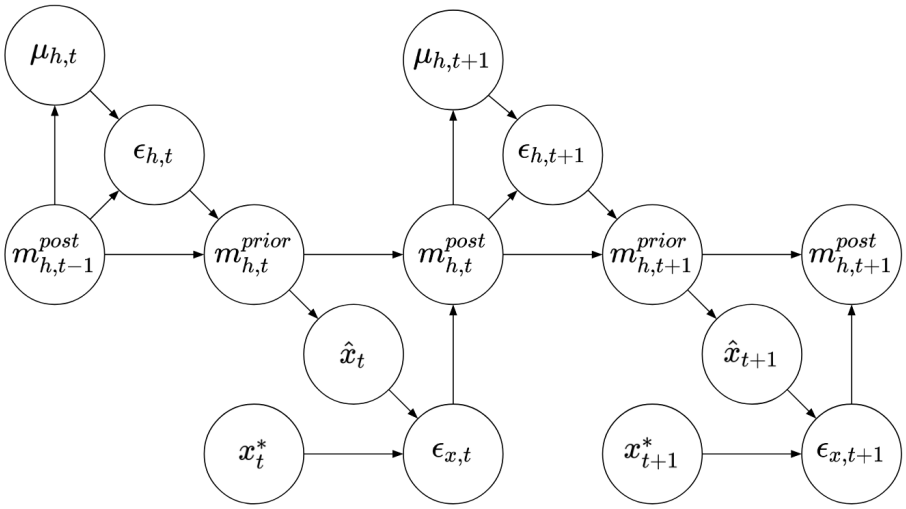


Figure 3.7: Simple PC-RNN computational graph.

This neural network is represented in figure 3.6b. In this representation, the intermediate variable $\hat{\boldsymbol{x}}_t$ is not represented to match the standard type of PC network, with error neurons and representation neurons interacting at each layer.

Figure 3.7 provides a computational model for this recurrent network. We can see that the same layer of neurons $\boldsymbol{m}_h$ receives several updates during what we have defined as one time step. This is made clearer in the computational model as we have defined two distinct variables $\boldsymbol{m}_h^{prior}$, computed based solely on top-down information from the prior density $p(\boldsymbol{H})$, and $\boldsymbol{m}_h^{post}$ merging in bottom-up information from the output layer.

By taking a closer look on the equations, we notice that this model can be simplified. Indeed, equations 3.32, 3.33, 3.34 can be merged into:

$$\boldsymbol{m}_{h,t}^{prior} = (1 - \frac{\alpha}{\tau\sigma_h^2})\boldsymbol{m}_{h,t-1}^{post} + \frac{\alpha}{\tau\sigma_h^2}\boldsymbol{W}_r \cdot \tanh\left(\boldsymbol{m}_{h,t-1}^{post}\right) \tag{3.38}$$

This update rule is very similar to that of an sRNN. In fact, if we do not have any prediction error on the output layer, i.e. if $\boldsymbol{\epsilon}_x = 0$, the model described by this set of equations is exactly the TRNN model presented in section 2.1.2, with a time constant of $\tau' = \frac{\tau\sigma_h^2}{\alpha}$.

Figure 3.8 provides a representation of the computational model with this shortcut, where the direct connection from $\boldsymbol{m}_{h,t-1}^{post}$ to $\boldsymbol{m}_{h,t}^{prior}$ now implements equation 3.38.

The pseudo-code performing a forward pass in the simple PC-RNN model is given in algorithm 2. This algorithm has four hyperparameters: τ, σ_x, σ_h, and α. Though, we can see from the equations that they only intervene through the two coefficients $\frac{\alpha}{\sigma_x^2}$ and $\frac{\alpha}{\tau\sigma_h^2}$. Consequently, we only consider those two coefficients when studying the influence of the model hyperparameters.

The proposed model can be separated into two distinct pathway. The generative, or top-down pathway, is formed by equations 3.38 and 3.35. The inference, or bottom-up pathway, is formed by equations 3.36 and 3.37.

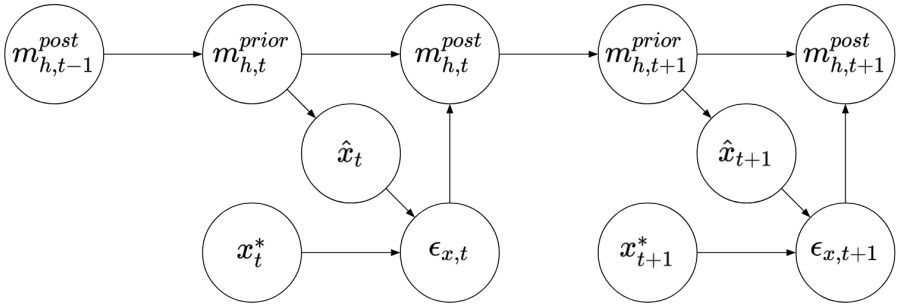


Figure 3.8: Simple PC-RNN simplified computational graph.

Algorithm 2: Forward pass through the simple PC-RNN model.

Parameters: $\boldsymbol{W}_o, \boldsymbol{W}_r, T, \tau, \sigma_x^2, \sigma_h^2, \alpha$
Inputs: $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$, $\boldsymbol{m}_{h,0}^{post}$
for $1 \leq t \leq T$ **do**
$\quad \boldsymbol{m}_{h,t}^{prior} \leftarrow (1 - \frac{\alpha}{\tau\sigma_h^2})\boldsymbol{m}_{h,t-1}^{post} + \frac{\alpha}{\tau\sigma_h^2}\boldsymbol{W}_r \cdot \tanh\left(\boldsymbol{m}_{h,t-1}^{post}\right)$;
$\quad \hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{W}_o \cdot \tanh\left(\boldsymbol{m}_{h,t}^{prior}\right)$;
$\quad \boldsymbol{\epsilon}_{x,t} \leftarrow \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t$;
$\quad \boldsymbol{m}_{h,t}^{post} \leftarrow \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\left(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\right) \odot \left(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\right)$;
end

Learning in this model can be performed in two ways. First, we can simply train this model using the BPTT algorithm. Second, as we have seen in section 2.2.3, the FEP also provides update rules for model parameters. If we also minimize VFE by a gradient descent on the parameters $\boldsymbol{W}_o$ and $\boldsymbol{W}_r$, we obtain the following learning rules:

$$\boldsymbol{W}_o \leftarrow \boldsymbol{W}_o + \frac{\lambda_o}{\sigma_x^2}\boldsymbol{\epsilon}_{x,t} \cdot \tanh(\boldsymbol{m}_{h,t}^{prior})^\intercal \tag{3.39}$$

$$\boldsymbol{W}_r \leftarrow \boldsymbol{W}_r + \frac{\lambda_r}{\tau\sigma_h^2}\boldsymbol{\epsilon}_{h,t+1} \cdot \tanh(\boldsymbol{m}_{h,t}^{post})^\intercal \tag{3.40}$$

However, we can already argue that PC-based learning is not able to achieve the same results as BPTT. We have seen in section 2.2.3 that this learning method can only approximate the BP algorithm if we let the inference process the time to converge. In our setting, inference is performed on a hidden variable with temporal dynamics, and thus never converges. Additionally, to prove the convergence, we have supposed that inference and learning were performed through the complete computational model. Here we only consider inference and learning through the computational model limited at one time-step, without any temporal unrolling.

After performing an hyperparameter search, we have experimentally compared the two learning methods. To perform this comparison, we have to take into account several factors. First, the BPTT algorithm is computationally more expensive than the PC-based learning. In term of temporal complexity, the BPTT needs to perform a backward run through the complete computational model, while in PC-based learning, the updates can be computed under way when performing the forward pass in the computational model. This suggests that a BPTT update is basically twice as expensive as a PC-based update. Second, there are several optimization algorithm that we can use with BPTT. To account for these two factors, we have compared the PC-based learning algorithm with twice as many updates than BPTT, and only used vanilla stochastic gradient descent for BPTT. The results, presented in figure 3.9, confirm our prediction that BPTT outperforms PC-based learning. More details about the experimental set up for this type of comparative study are provided in section 3.4. We present these results in advance as a reason for not providing the derivations of the PC-based learning rules in each of the following models, although they could be derived in a similar fashion.

In conclusion, we have seen how minimization of VFE on a simple model can lead, with some assumptions, to a set of equations aligning with TRNNs, with additional bottom-up inference

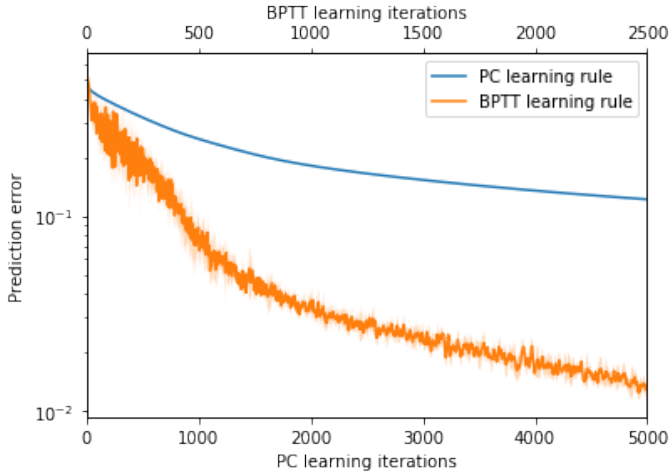


Figure 3.9: Comparison between the PC-based learning algorithm and BPTT. These results were obtained one a data set of 64 trajectories, with the following parameters: $d_h = 50$, $d_o = 2$, $\frac{\alpha}{\sigma_x^2} = 0.001$, $\frac{\alpha}{\tau\sigma_h^2} = 0.1, \frac{\lambda_r}{\sigma_x^2} = 3$ and $\frac{\lambda_o}{\tau\sigma_h^2} = 2$. The represented learning curves correspond to an average computed over 5 random seeds.

connections. We have also seen that although this method also dictates update rules for model parameters, it is preferable to use the BPTT algorithm for learning.

This model is very similar to the model presented in (Ororbia et al., 2020). The main difference lies in the fact that in our model, the weights of the feedback connections are not learned and directly depend on the weights of the feedforward connections. These weights should be better suited for inference, as this directly implements gradient descent on the prediction error.

To derive this model, we have used the "trick" of making $\boldsymbol{\mu}_h$ dependent on the past value of $\boldsymbol{m}_h$. Without this dependency, the resulting model would correspond to a one-layer feedforward network augmented with feedback connections. This type of model would not be able to properly predict a temporal signal $\boldsymbol{x}_t^*$. In the FEP literature however, dynamics are brought via the notion of generalized coordinates, we try in the next section to derive an RNN model from this principle.

3.3.2 Using first-order generalized coordinates

Another way of implementing a dynamical generative model is to have a multivariate random variable responsible for each order of motion (velocity, acceleration, jerk, etc). In our case, we limit ourselves to using a random variable for the the first-order derivative of the hidden state, that we call $\boldsymbol{H}'$. The corresponding probabilistic model is represented in figure 3.10a.

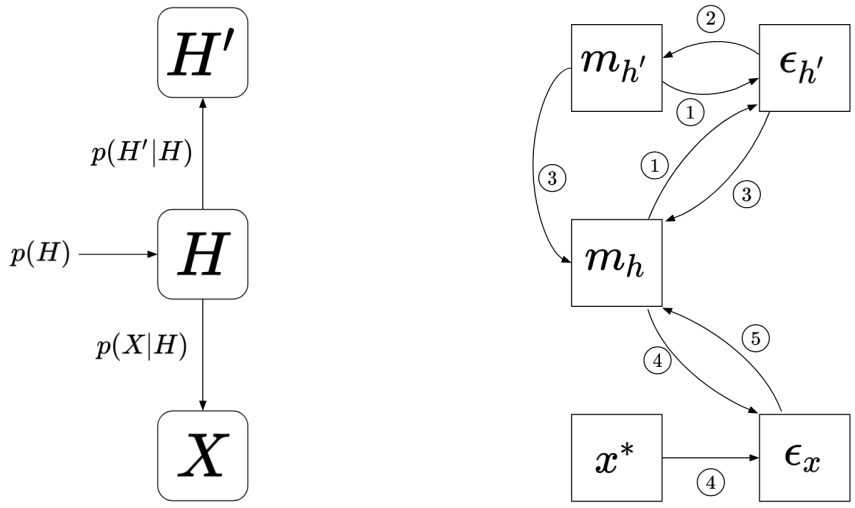
The generative model is given by the following set of equations:

$$p(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{\mu}_h, \sigma_h^2 \mathbb{I}_{d_h}) \tag{3.41}$$
$$p(\boldsymbol{h}'|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}'; \boldsymbol{f}(\boldsymbol{h}), \sigma_{h'}^2 \mathbb{I}_{d_h}) \tag{3.42}$$
$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{g}(\boldsymbol{h}), \sigma_x^2 \mathbb{I}_{d_x}) \tag{3.43}$$

In comparison with the previous model, we have added the Gaussian density $p(\boldsymbol{h}'|\boldsymbol{h})$ of mean $\boldsymbol{f}(\boldsymbol{h})$ and variance $\sigma_{h'}^2$. The functions $\boldsymbol{f}$ and $\boldsymbol{g}$ are defined as follows:

$$\boldsymbol{f}(\boldsymbol{h}) = \frac{1}{\tau}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{h}) - \boldsymbol{h}\big) \tag{3.44}$$
$$\boldsymbol{g}(\boldsymbol{h}) = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{h}) \tag{3.45}$$

To perform variational inference, we introduce the recognition density $q(\boldsymbol{H}, \boldsymbol{H}')$:



(a) (b)

Figure 3.10: PC-RNN with generalized coordinates probabilistic model and neural network representation. **Left:** Graphical representation of the generative model. **Right:** Neural network representation of the resulting FE minimization process. The synaptic connections are numbered according to the order in which we perform the corresponding computations during a single time step.

$$q(\boldsymbol{h}, \boldsymbol{h'}) = q(\boldsymbol{h})q(\boldsymbol{h'}) \tag{3.46}$$
$$q(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{m}_h, v_h \mathbb{I}_{d_h}) \tag{3.47}$$
$$q(\boldsymbol{h'}) = \mathcal{N}(\boldsymbol{h'}; \boldsymbol{m}_{h'}, v_{h'} \mathbb{I}_{d_h}) \tag{3.48}$$

$\boldsymbol{H}$ and $\boldsymbol{H'}$ are assumed to be independent according to this recognition density, which thus factors into two marginal density functions $q(\boldsymbol{H})$ and $q(\boldsymbol{H'})$. Both marginal distributions are assumed to be Gaussian, respectively with means $\boldsymbol{m}_h$ and $\boldsymbol{m}_{h'}$, and variances v_h and $v_{h'}$. We can write the VFE as:

$$F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}) = E(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}) + C \tag{3.49}$$

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}) = & \frac{1}{2\sigma_x^2} \|\boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)\|_2^2 + \frac{d_x}{2} \log(\sigma_x^2) \\ & + \frac{1}{2\sigma_h^2} \|\boldsymbol{m}_h - \boldsymbol{\mu}_h\|_2^2 + \frac{d_h}{2} \log(\sigma_h^2) \\ & + \frac{1}{2\sigma_{h'}^2} \|\boldsymbol{m}_{h'} - \boldsymbol{f}(\boldsymbol{m}_h)\|_2^2 + \frac{d_h}{2} \log(\sigma_{h'}^2) \\ & + C \end{aligned} \tag{3.50}$$

We can now derive from this expression the gradient of the VFE with regard to the recognition density parameters $\boldsymbol{m}_h$ and $\boldsymbol{m}_{h'}$. We start with $\boldsymbol{m}_{h'}$ and denote by $\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'})}{\partial m_{h',i}}$ the partial derivative with regard to the coefficient i of $\boldsymbol{m}_{h'}$.

$$\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'})}{\partial m_{h',i}} = \frac{1}{\sigma_{h'}^2} \big(m_{h',i} - f_i(\boldsymbol{m}_h)\big) \tag{3.51}$$

where $f_i(\boldsymbol{m}_h)$ denotes the i-th component of $\boldsymbol{f}(\boldsymbol{m}_h)$.

This equation is quite simple, and basically states that minimization of the free-energy should

pull $\boldsymbol{m}_{h'}$ towards the prediction $\boldsymbol{f}(\boldsymbol{m}_h)$. This leads to the following update rule for $\boldsymbol{m}_{h'}$:

$$\boldsymbol{m}_{h'} \leftarrow \boldsymbol{m}_{h'} - \underbrace{\frac{\alpha}{\sigma_{h'}^2}\Big(\boldsymbol{m}_{h'} - \frac{1}{\tau}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}_h) - \boldsymbol{m}_h\big)\Big)}_{\text{Top-down}} \tag{3.52}$$

We follow-up by the gradient computation with regard to the i-th coefficient of $\boldsymbol{m}_h$:

$$\begin{aligned}\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'})}{\partial m_{h,i}} =& -\frac{1}{\sigma_x^2}\sum_{j=1}^{d_x}\big(x_j^* - g_j(\boldsymbol{m}_h)\big)\frac{\partial g_j(\boldsymbol{m}_h)}{\partial m_{h,i}} \\ &+ \frac{1}{\sigma_h^2}\big(m_{h,i} - \mu_{h,i}\big) \\ &+ \frac{1}{\sigma_{h'}^2}\sum_{j=1}^{d_h}\big(m_{h',j} - f_j(\boldsymbol{m}_h)\big)\frac{\partial f_j(\boldsymbol{m}_h)}{\partial m_{h,i}}\end{aligned} \tag{3.53}$$

By injecting the definitions of $\boldsymbol{f}$ and $\boldsymbol{g}$ into this equation, we obtain:

$$\begin{aligned}\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'})}{\partial m_{h,i}} =& -\frac{1}{\sigma_x^2}\sum_{j=1}^{d_x}\big(x_j^* - g_j(\boldsymbol{m}_h)\big)\big(1 - \tanh^2(m_{h,i})\big)W_o^{ij} \\ &+ \frac{1}{\sigma_h^2}\big(m_{h,i} - \mu_{h,i}\big) \\ &+ \frac{1}{\tau\sigma_{h'}^2}\sum_{j=1}^{d_h}\big(m_{h',j} - f_j(\boldsymbol{m}_h)\big)\Big(W_r^{ji}\big(1 - \tanh^2(m_{h,i})\big) - \mathbb{I}_{d_h}^{ij}\Big)\end{aligned} \tag{3.54}$$

Which results in the following update rule for $\boldsymbol{m}_h$:

$$\begin{aligned}\boldsymbol{m}_h \leftarrow \boldsymbol{m}_h &\underbrace{+\frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_h)\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_x\big)}_{\text{Bottom-up from } X} \\ &\underbrace{-\frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_h}_{\text{Top-down}} \\ &\underbrace{+\frac{\alpha}{\tau\sigma_{h'}^2}\Big(\big(1 - \tanh^2(\boldsymbol{m}_h)\big) \odot \boldsymbol{W}_r^\intercal - \mathbb{I}_{d_h}\Big) \cdot \boldsymbol{\epsilon}_{h'}}_{\text{Bottom-up from } H'} \\ &\underbrace{+\boldsymbol{m}_{h'}}_{\text{Velocity}}\end{aligned} \tag{3.55}$$

The first added term in this update rule comes from the output layer, similarly to the previous model. This term pulls $\boldsymbol{m}_h$ towards a value that decreases the prediction error $\boldsymbol{\epsilon}_x$.

The second added term corresponds to the minimization of the discrepancy between the prior density mean $\boldsymbol{\mu}_h$ and the recognition density mean $\boldsymbol{m}_h$. It pulls $\boldsymbol{m}_h$ towards $\boldsymbol{\mu}_h$. In this model, we find that this term is of little interest. To remove it, we can simply consider that σ_h^2 is large enough for this term to become negligible. Intuitively, this corresponds to having a very flat prior distribution on $\boldsymbol{H}$, thus not impacting much when computing the VFE. In our implementation of this model, we overlook this term.

The third added term is probably the least intuitive. This term comes from the h' layer, and pulls $\boldsymbol{m}_h$ towards a value that decreases the prediction error $\boldsymbol{\epsilon}_{h'}$.

Finally, the last term of this update rule, $\boldsymbol{m}_{h'}$, comes from the definition of $\boldsymbol{H}'$ as the first-order derivative of $\boldsymbol{H}$. The dynamics of $\boldsymbol{m}_h$ are guided both by the gradient descent on VFE and this first-order derivative.

From equations 3.52 and 3.55, we can construct a set of equations guiding the temporal evolution of the variables of our neural network:

$$\hat{\boldsymbol{m}}_{h',t} = \frac{1}{\tau}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}^{post}_{h,t-1}) - \boldsymbol{m}^{post}_{h,t-1}\big) \tag{3.56}$$

$$\boldsymbol{\epsilon}_{h',t} = \boldsymbol{m}_{h',t-1} - \hat{\boldsymbol{m}}_{h',t} \tag{3.57}$$

$$\boldsymbol{m}_{h',t} = \boldsymbol{m}_{h',t-1} - \frac{\alpha}{\sigma^2_{h'}}\boldsymbol{\epsilon}_{h',t} \tag{3.58}$$

$$\boldsymbol{m}^{prior}_{h,t} = \boldsymbol{m}^{post}_{h,t-1} + \boldsymbol{m}_{h',t} + \frac{\alpha}{\tau\sigma^2_{h'}}\Big(\big(1 - \tanh^2(\boldsymbol{m}^{post}_{h,t-1})\big) \odot \boldsymbol{W}_r^\intercal - \mathbb{I}_{d_h}\Big) \cdot \boldsymbol{\epsilon}_{h',t} \tag{3.59}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{m}^{prior}_{h,t}) \tag{3.60}$$

$$\boldsymbol{\epsilon}_{x,t} = \boldsymbol{x}^*_t - \hat{\boldsymbol{x}}_t \tag{3.61}$$

$$\boldsymbol{m}^{post}_{h,t} = \boldsymbol{m}^{prior}_{h,t} + \frac{\alpha}{\sigma^2_x}\big(1 - \tanh^2(\boldsymbol{m}^{prior}_{h,t})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big) \tag{3.62}$$

This neural network is represented in figure 3.10b. In this representation, the intermediate variables $\hat{\boldsymbol{x}}_t$ and $\hat{\boldsymbol{m}}_{h,t}$ are not represented to match the standard type of PC network, with error neurons and representation neurons interacting at each layer.

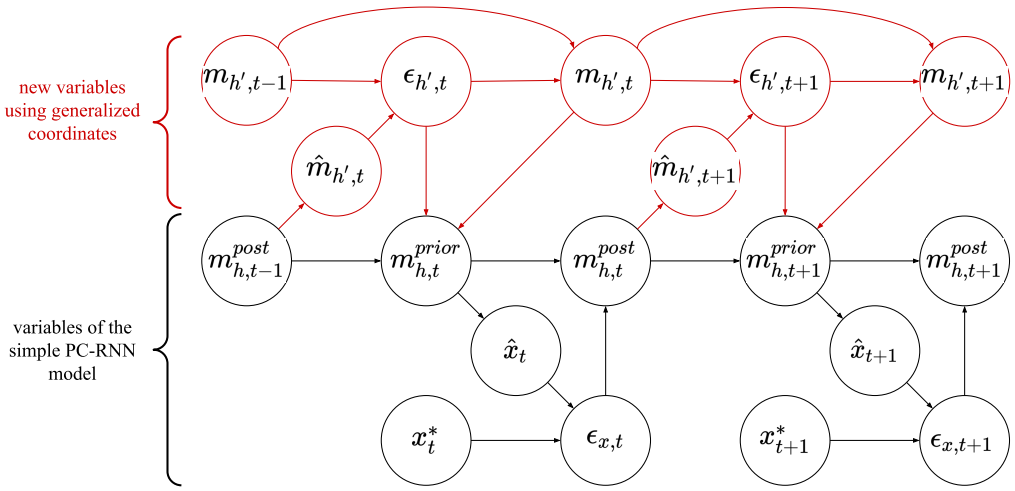


Figure 3.11: PC-RNN with generalized coordinates computational graph. We represent in red the variables added by the use of generalized coordinates.

Figure 3.11 provides a clearer computational model for this recurrent network. This network is quite intricate and the influence of the additional layer maintaining through time an estimation of the velocity of the hidden states, is not clear.

One observation we can make is that this model seems to relate to the former model under some conditions. Indeed, if we remove the last term of equation 3.59, and assume that $\frac{\alpha}{\sigma^2_{h'}} = 1$, we can merge equations 3.56, 3.57, 3.58 and 3.59 into:

$$\boldsymbol{m}^{prior}_{h,t} = (1 - \frac{1}{\tau})\boldsymbol{m}^{post}_{h,t-1} + \frac{1}{\tau}\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}^{post}_{h,t-1}) \tag{3.63}$$

Without these assumptions, the complete model performs some additional computations. We could basically describe these computations as follows. First, the model maintains through time an estimation of the velocity of the hidden layer. At each time step, this estimation is pulled towards the prediction $\hat{\boldsymbol{m}}_{h',t}$. On the other hand, the discrepancy between this prediction and the current value of the velocity acts as a prediction error that the hidden layer tries to minimize through feedback connections. We studied the effect of this precise inference mechanism, and our results indicate that removing it does not decrease the ability of the model to encode a large number of patterns, while improving its computational speed.

The pseudo-code performing a forward pass in the PC-RNN model with generalized coordinates is given in algorithm 3.

Algorithm 3: Forward pass through the PC-RNN model with generalized coordinates.

Parameters: $\boldsymbol{W}_o, \boldsymbol{W}_r, T, \tau, \sigma_x^2, \sigma_{h'}^2, \alpha$
Inputs: $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$, $\boldsymbol{m}_{h,0}^{post}$, $\boldsymbol{m}_{h',0}$
for $1 \leq t \leq T$ **do**

$\hat{\boldsymbol{m}}_{h',t} \leftarrow \frac{1}{\tau}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post}) - \boldsymbol{m}_{h,t-1}^{post}\big)$;
$\boldsymbol{\epsilon}_{h',t} \leftarrow \boldsymbol{m}_{h',t-1} - \hat{\boldsymbol{m}}_{h',t}$;
$\boldsymbol{m}_{h',t} \leftarrow \boldsymbol{m}_{h',t-1} - \frac{\alpha}{\sigma_{h'}^2}\boldsymbol{\epsilon}_{h',t}$;
$\boldsymbol{m}_{h,t}^{prior} \leftarrow \boldsymbol{m}_{h,t-1}^{post} + \boldsymbol{m}_{h',t} + \frac{\alpha}{\tau\sigma_{h'}^2}\Big(\big(1 - \tanh^2(\boldsymbol{m}_{h,t-1}^{post})\big) \odot \boldsymbol{W}_r^\intercal - \mathbb{I}_{d_h}\Big) \cdot \boldsymbol{\epsilon}_{h',t}$;
$\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{W}_o \cdot \tanh\big(\boldsymbol{m}_{h,t}^{prior}\big)$;
$\boldsymbol{\epsilon}_{x,t} \leftarrow \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t$;
$\boldsymbol{m}_{h,t}^{post} \leftarrow \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big)$;

end

In these two sections, we have derived RNN models according to the FEP. Contrary to sRNNs, these models are capable of performing online inference of their hidden state. However, they are not able to infer their initial hidden state based on the complete trajectory. As explained in the introduction of this chapter, being able to infer the *cause* of a trajectory would be an interesting feature.

In the next section, we build a model that tries to include such considerations into its design.

3.3.3 Simple recurrent model with hidden causes

The problem with the previous models is that they only perform inference on a dynamic variable $\boldsymbol{H}$. This variational inference mechanism allows adapting the estimation of $\boldsymbol{H}$ at each time step, based on the prediction error on the output level. But since $\boldsymbol{H}$ is subject to temporal dynamics other than that caused by the variational inference mechanism, it is impossible to infer the initial value of $\boldsymbol{H}$. This initial value can be considered as the *cause* of the trajectory, as it can condition the whole generative process.

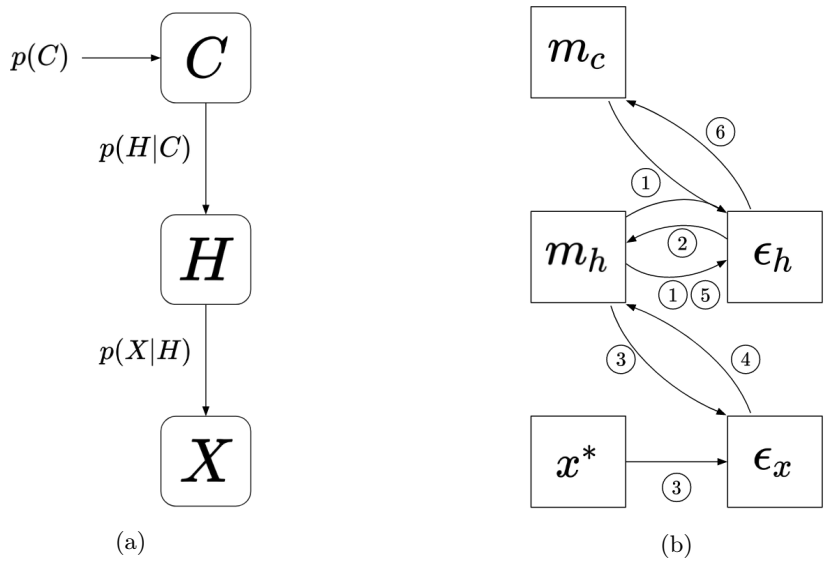


Figure 3.12: PC-RNN with hidden causes probabilistic model and neural network representation. **Left:** Graphical representation of the generative model. **Right:** Neural network representation of the resulting FE minimization process. The synaptic connections are numbered according to the order in which we perform the corresponding computations during a single time step.

To design an RNN model with *causes* that can be inferred dynamically, we add a new multivariate random variable influencing the temporal dynamics of $\boldsymbol{H}$. To align with the FEP literature,

we name this variable hidden causes, and denote it as $\boldsymbol{C}$. The corresponding probabilistic model is represented in figure 3.12a. It is described by the following set of equations:

$$p(\boldsymbol{c}) = \mathcal{N}(\boldsymbol{c}; \boldsymbol{\mu}_c, \sigma_c^2 \mathbb{I}_{d_c}) \tag{3.64}$$
$$p(\boldsymbol{h}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{f}(\boldsymbol{c}, \boldsymbol{h}_{past}), \sigma_h^2 \mathbb{I}_{d_h}) \tag{3.65}$$
$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(x; \boldsymbol{g}(\boldsymbol{h}), \sigma_x^2 \mathbb{I}_{d_x}) \tag{3.66}$$

Similarly to the first presented model, we use a "trick" to enforce temporal dynamics onto the variable $\boldsymbol{H}$. While in the first model, the future value of $\boldsymbol{H}$ only depended on its past value, here it also depends on the hidden causes variable $\boldsymbol{C}$. To perform variational inference, we introduce the recognition density function $q(\boldsymbol{H}, \boldsymbol{C})$:

$$q(\boldsymbol{h}, \boldsymbol{c}) = q(\boldsymbol{h})q(\boldsymbol{c}) \tag{3.67}$$
$$q(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{m}_h, v_h \mathbb{I}_{d_h}) \tag{3.68}$$
$$q(\boldsymbol{c}) = \mathcal{N}(\boldsymbol{c}; \boldsymbol{m}_c, v_c \mathbb{I}_{d_c}) \tag{3.69}$$

H and C are assumed to be independent according to this recognition density, which thus factors into two marginal density functions $q(\boldsymbol{H})$ and $q(\boldsymbol{C})$. Both marginal distributions are assumed to be Gaussian, respectively with means $\boldsymbol{m}_h$ and $\boldsymbol{m}_c$, and variances v_h and v_c. We can write the VFE as:

$$F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c) = E(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c) + C \tag{3.70}$$

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c) = & \frac{1}{2\sigma_x^2} \|\boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)\|_2^2 + \frac{d_x}{2} \log(\sigma_x^2) \\ & + \frac{1}{2\sigma_h^2} \|\boldsymbol{m}_h - \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})\|_2^2 + \frac{d_h}{2} \log(\sigma_h^2) \\ & + \frac{1}{2\sigma_c^2} \|\boldsymbol{m}_c - \boldsymbol{\mu}_c\|_2^2 + \frac{d_c}{2} \log(\sigma_c^2) \\ & + C \end{aligned} \tag{3.71}$$

We can derive from this expression the gradient of the VFE with regard to the recognition distribution parameters $\boldsymbol{m}_h$ and $\boldsymbol{m}_c$:

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c)}{\partial m_{h,i}} = & -\frac{1}{\sigma_x^2} \sum_{j=1}^{d_x} \big(x_j^* - g_j(\boldsymbol{m}_h)\big) \frac{\partial g_j(\boldsymbol{m}_h)}{\partial m_{h,i}} \\ & + \frac{1}{\sigma_h^2} \big(m_{h,i} - f_i(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big) \end{aligned} \tag{3.72}$$

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c)}{\partial m_{c,i}} = & -\frac{1}{\sigma_h^2} \sum_{j=1}^{d_h} \big(m_{h,j} - f_j(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big) \frac{\partial f_j(\boldsymbol{m}_c, \boldsymbol{h}_{past})}{\partial m_{c,i}} \\ & + \frac{1}{\sigma_c^2} (m_{c,i} - \mu_{c,i}) \end{aligned} \tag{3.73}$$

The second term of this last equation pulls $\boldsymbol{m}_c$ towards the prior distribution mean $\boldsymbol{\mu}_c$. If we consider this prior to be of very low precision, i.e. very high variance, this term becomes negligible and does not affect the rest of the computations. In the model presented here, we consider that this prior distribution on $\boldsymbol{C}$ is of little interest and thus we omit this term in the following derivations. In chapter 5, we study ways to exploit this prior distribution for memory retrieval.

Removing this term, and using the same definition for $\boldsymbol{g}$ (equation 3.45), we can obtain the

following update rules for $\boldsymbol{m}_h$ and $\boldsymbol{m}_c$:

$$\begin{aligned}\boldsymbol{m}_h \leftarrow \boldsymbol{m}_h &\underbrace{+\frac{\alpha}{\sigma_x^2}\big(1-\tanh^2(\boldsymbol{m}_h)\big)\odot\big(\boldsymbol{W}_o^\intercal\cdot\boldsymbol{\epsilon}_x\big)}_{\text{Bottom-up}} \\ &\underbrace{-\frac{\alpha}{\sigma_h^2}\big(\boldsymbol{m}_h-\boldsymbol{f}(\boldsymbol{m}_c,\boldsymbol{m}_h)\big)}_{\text{Top-down}}\end{aligned} \tag{3.74}$$

$$\boldsymbol{m}_c \leftarrow \boldsymbol{m}_c \underbrace{+\frac{\alpha}{\sigma_h^2}\big(\boldsymbol{m}_h-\boldsymbol{f}(\boldsymbol{m}_c,\boldsymbol{m}_h)\big)\cdot\nabla_{\boldsymbol{m}_c}\boldsymbol{f}(\boldsymbol{m}_c,\boldsymbol{m}_h)}_{\text{Bottom-up}} \tag{3.75}$$

These two update rules depend on the function $\boldsymbol{f}$ that we still have not defined. We consider two possible implementations. The first possibility is to use the additional variable $\boldsymbol{C}$ as an additive influence on the temporal dynamics of $\boldsymbol{H}$, as described by this equation:

$$f_{add}(\boldsymbol{m}_c,\boldsymbol{m}_h)=(1-\frac{1}{\tau})\boldsymbol{m}_h+\frac{1}{\tau}\big(\boldsymbol{W}_r\cdot\tanh(\boldsymbol{m}_h)+\boldsymbol{W}_c\cdot\boldsymbol{m}_c\big) \tag{3.76}$$

In this implementation, the term $\boldsymbol{W}_c\cdot\boldsymbol{m}_c$ can be seen as an additive bias in the recurrent update of the RNN. The matrix $\boldsymbol{W}_c$ of dimension (d_h,d_c) can be seen as a basis of possible biases, and the hidden causes $\boldsymbol{m}_c$ as mixing coefficient to select a bias in this vector space. Reproducing this idea on the recurrent weights $\boldsymbol{W}_r$ led us to the second implementation, described by the equation:

$$f_{mult}(\boldsymbol{m}_c,\boldsymbol{m}_h)=(1-\frac{1}{\tau})\boldsymbol{m}_h+\frac{1}{\tau}\big((\boldsymbol{W}_R\cdot\boldsymbol{m}_c)\cdot\tanh(\boldsymbol{m}_h)\big) \tag{3.77}$$

where $\boldsymbol{W}_R$ denotes a tensor of dimension (d_h,d_h,d_c) that can be seen as a basis of size d_c of possible recurrent weights. Again, the hidden causes $\boldsymbol{m}_c$ act as mixing coefficient to select the recurrent weights of the model. The idea of conditioning the weight matrix with an external input was first introduced with the Multiplicative RNN model (Sutskever et al., 2011), where it was combined with hessian free optimization.

To avoid scaling issues when dealing with three-way tensors, it is proposed in (Taylor and Hinton, 2009) to factor it into three matrices, such that for all i,j,k, $W_R^{ijk}=\sum_{l<d_f}W_p^{il}\cdot W_f^{jl}\cdot W_c^{kl}$. The model can scale better to large hidden state and hidden causes dimensions with this factorization, since the factor dimension d_f can be adjusted to control the number of model parameters. In our experiments, we always use $d_f=d_h/2$.

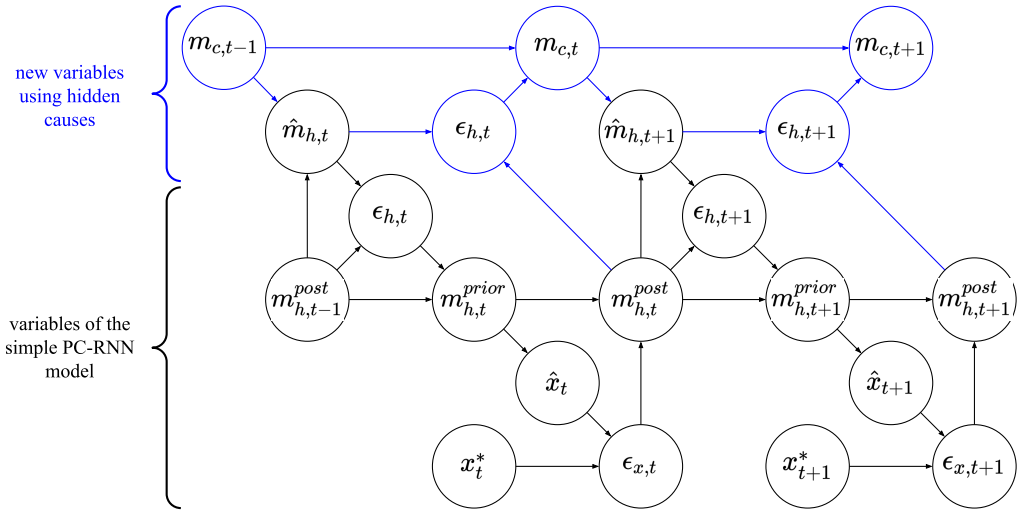


Figure 3.13: PC-RNN with hidden causes computational graph. We represent in blue the variables added by the use of hidden causes.

Both implementations have the same neural network representation, displayed in figure 3.12b. Note that in this figure the connection from $\boldsymbol{m}_h$ to $\boldsymbol{\epsilon}_h$ is annotated twice. This is because we

compute the variable $\boldsymbol{\epsilon}_h$ twice, based on two different versions of the variable $\boldsymbol{m}_h$. This is made clearer with the full computational model provided in figure 3.13. Intuitively, updating the value of $\boldsymbol{\epsilon}_h$ under way provides a better error signal for the inference of $\boldsymbol{C}$ (i.e. the update of $\boldsymbol{m}_c$), since this new value incorporates information from the bottom-up signal originating from the output layer.

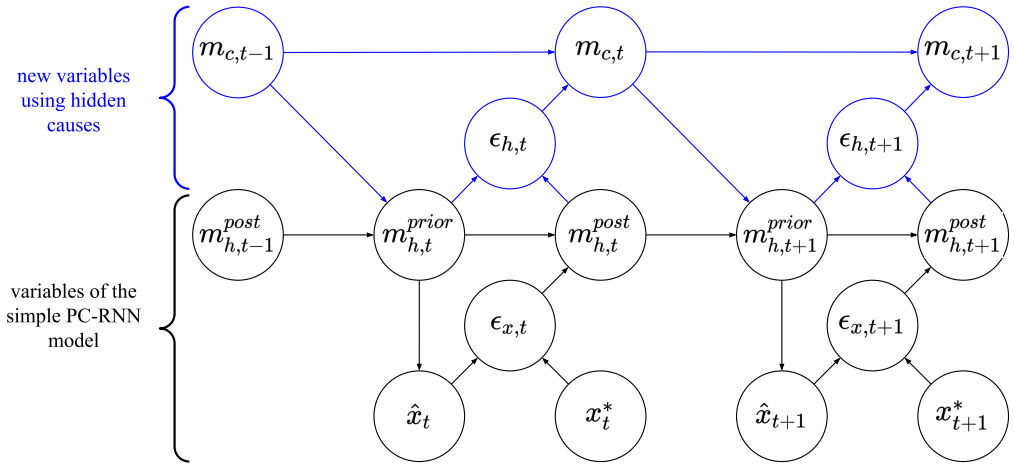


Figure 3.14: PC-RNN with hidden causes simplified computational graph. We represent in blue the variables added by the use of hidden causes.

We can also operate the same simplification on the computational graph that we have used on the first model. By merging some computations into a single update, we obtain the simplified computational graph represented in figure 3.14. On this figure, we have also rearranged the position of some of the variables to highlight the parallelism of the inference processes. The prediction error on the output layer $\boldsymbol{\epsilon}_x$ is used to adjust the hidden states belief $\boldsymbol{m}_h$, in the same way that the prediction error on the hidden states $\boldsymbol{\epsilon}_h$ is used to adjust the hidden causes belief $\boldsymbol{m}_c$.

The additive hidden causes computational model can be described with the following set of equations:

$$\boldsymbol{m}^{prior}_{h,t} = (1 - \frac{\alpha}{\tau\sigma_h^2})\boldsymbol{m}^{post}_{h,t-1} + \frac{\alpha}{\tau\sigma_h^2}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}^{post}_{h,t-1}) + \boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1}\big) \tag{3.78}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{m}^{prior}_{h,t}) \tag{3.79}$$

$$\boldsymbol{\epsilon}_{x,t} = \boldsymbol{x}^*_t - \hat{\boldsymbol{x}}_t \tag{3.80}$$

$$\boldsymbol{m}^{post}_{h,t} = \boldsymbol{m}^{prior}_{h,t} + \frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}^{prior}_{h,t})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big) \tag{3.81}$$

$$\boldsymbol{\epsilon}_{h,t} = \boldsymbol{m}^{post}_{h,t} - \boldsymbol{m}^{prior}_{h,t} \tag{3.82}$$

$$\boldsymbol{m}_{c,t} = \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_h^2}\boldsymbol{W}_c^\intercal \cdot \boldsymbol{\epsilon}_{h,t} \tag{3.83}$$

The multiplicative hidden causes model is described by the same equations, except for equations 3.78 and 3.83 that are respectively replaced by:

$$\boldsymbol{m}^{prior}_{h,t} = (1 - \frac{\alpha}{\tau\sigma_h^2})\boldsymbol{m}^{post}_{h,t-1} + \frac{\alpha}{\tau\sigma_h^2}\boldsymbol{W}_f^\intercal \cdot \Big(\big(\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}^{post}_{h,t-1})\big) \odot (\boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1})\Big) \tag{3.84}$$

$$\boldsymbol{m}_{c,t} = \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_h^2}\Big(\big(\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}^{post}_{h,t-1})\big) \odot \boldsymbol{W}_c\Big)^\intercal \cdot \boldsymbol{W}_f \cdot \boldsymbol{\epsilon}_{h,t} \tag{3.85}$$

The pseudo-code performing a forward pass in the PC-RNN model with additive hidden causes is given in algorithm 4. It can be easily adapted for the multiplicative case.

The main interest of these two models is that the random variable $\boldsymbol{C}$, that causally affects the complete trajectory, can be inferred dynamically. This feature can be exploited in several ways:

Algorithm 4: Forward pass through the PC-RNN model with additive hidden causes.

Parameters: $\boldsymbol{W}_o, \boldsymbol{W}_r, \boldsymbol{W}_c, T, \tau, \sigma_x^2, \sigma_h^2, \alpha$
Inputs: $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$, $\boldsymbol{m}_{h,0}^{post}$, $\boldsymbol{m}_{c,0}$
for $1 \leq t \leq T$ **do**

$\boldsymbol{m}_{h,t}^{prior} \leftarrow (1 - \frac{\alpha}{\tau\sigma_h^2})\boldsymbol{m}_{h,t-1}^{post} + \frac{\alpha}{\tau\sigma_h^2}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post}) + \boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1}\big)$;
$\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{W}_o \cdot \tanh\big(\boldsymbol{m}_{h,t}^{prior}\big)$;
$\boldsymbol{\epsilon}_{x,t} \leftarrow \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t$;
$\boldsymbol{m}_{h,t}^{post} \leftarrow \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big)$;
$\boldsymbol{\epsilon}_{h,t} \leftarrow \boldsymbol{m}_{h,t}^{post} - \boldsymbol{m}_{h,t}^{prior}$;
$\boldsymbol{m}_{c,t} \leftarrow \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_h^2}\boldsymbol{W}_c^\intercal \cdot \boldsymbol{\epsilon}_{h,t}$;

end

- First, our RNN can act both as a generative model (decoder) and as a representation model (encoder), starting from a random estimation of the hidden causes $\boldsymbol{m}_{c,0}$, and inferring a value of $\boldsymbol{C}$ that better matches a target trajectory. The input becomes $(\boldsymbol{x}_1^*, \ldots, \boldsymbol{x}_T^*)$, and the output becomes $\boldsymbol{m}_{c,T}$, which amounts to an inversion of the sequence memory function f.
- Second, this reversibility can be interesting in the context of lifelong learning. When confronted with new sequence patterns to learn, this system might be able to infer suitable hidden causes to model those patterns, without requiring any modification to the model parameters. Therefore, this could help to avoid the problem of catastrophic forgetting. This question is explored in the experimental section of this chapter.
- Finally, this reversibility can be interesting for the problem of memory retrieval, aiming to retrieve a learned pattern from a noisy or incomplete version of this pattern. Based on a distorted version of one of the learned trajectory patterns, the inference of $\boldsymbol{C}$ might provide appropriate values for the generation of the real pattern. This question is developed in chapter 5.

We can note that the latent variable that we call hidden causes is very similar to the notion of parametric bias in RNNs as presented in (Tani and Ito, 2003).

In some of the future experiments use learning rules derived from PC for these two models. The learning rule for the output weights $\boldsymbol{W}_o$ are the same as the ones for the first RNN model we proposed:

$$\boldsymbol{W}_o \leftarrow \boldsymbol{W}_o + \frac{\lambda_o}{\sigma_x^2}\boldsymbol{\epsilon}_{x,t} \cdot \tanh(\boldsymbol{m}_{h,t}^{prior})^\intercal \tag{3.86}$$

where λ_o is the learning rate associated with the weight matrix $\boldsymbol{W}_o$. This rule locally minimizes VFE by updating the model parameters in a direction that minimize the local prediction error $\boldsymbol{\epsilon}_x$. We can apply the same method to derive the learning rules on the weights $\boldsymbol{W}_r$ and $\boldsymbol{W}_c$ in the additive model:

$$\boldsymbol{W}_r \leftarrow \boldsymbol{W}_r + \frac{\lambda_r}{\tau\sigma_h^2}\boldsymbol{\epsilon}_{h,t+1} \cdot \tanh(\boldsymbol{m}_{h,t}^{post})^\intercal \tag{3.87}$$

$$\boldsymbol{W}_c \leftarrow \boldsymbol{W}_c + \frac{\lambda_c}{\tau\sigma_h^2}\boldsymbol{\epsilon}_{h,t+1} \cdot \boldsymbol{m}_{c,t}^\intercal \tag{3.88}$$

where λ_r and λ_c are the learning rates associated with the weight matrices $\boldsymbol{W}_r$ and $\boldsymbol{W}_c$. For the multiplicative model, the local gradient descent derivations are a bit more complex, and provide the following learning rules for the $\boldsymbol{W}_p$, $\boldsymbol{W}_f$ and $\boldsymbol{W}_c$:

$$\boldsymbol{W}_p \leftarrow \boldsymbol{W}_p + \frac{\lambda_p}{\tau\sigma_h^2} \tanh(\boldsymbol{m}_{h,t}^{post}) \cdot \left((\boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t}) \odot (\boldsymbol{W}_f \cdot \boldsymbol{\epsilon}_{h,t+1})\right)^\intercal \tag{3.89}$$

$$\boldsymbol{W}_f \leftarrow \boldsymbol{W}_f + \frac{\lambda_f}{\tau\sigma_h^2} \boldsymbol{\epsilon}_{h,t+1} \cdot \left((\boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t}) \odot (\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}_{h,t}^{post}))\right)^\intercal \tag{3.90}$$

$$\boldsymbol{W}_c \leftarrow \boldsymbol{W}_c + \frac{\lambda_c}{\tau\sigma_h^2} \boldsymbol{m}_{c,t} \cdot \left((\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}_{h,t}^{post})) \odot (\boldsymbol{W}_f \cdot \boldsymbol{\epsilon}_{h,t+1})\right)^\intercal \tag{3.91}$$

where λ_p and λ_f are the learning rates associated with the weight matrices $\boldsymbol{W}_p$ and $\boldsymbol{W}_f$. For each of our experimental set up, we specify whether learning is performed using BPTT or using these learning rules.

3.3.4 Combining first-order generalized coordinates and hidden causes

Motivated by the results we obtained with the first-order generalized coordinates model, and with the hidden causes model, we try in this section to combine both ideas.

The model we are going to propose here is not strictly aligned with the FEP, it is an attempt to merge in an efficient way the concepts of first-order generalized coordinates and hidden causes.

As we have seen in the three past models we have designed, integrating recurrent dynamics into the FEP can be quite tricky. In the first proposed model, and in the hidden causes model, we have hidden the recurrence in the dependency of the prior density mean $\boldsymbol{\mu}_h$ with regard to the past recognition mean $\boldsymbol{m}_h$. This dependency is not reflected in the probabilistic model, and thus no inference of the past recognition mean with regard to the current output prediction error is considered.

Resorting to generalized coordinates might seem like a proper solution, but in the inference process we ignore the fact that $\boldsymbol{H}$ directly depends on $\boldsymbol{H'}$. We can infer $\boldsymbol{H}$ from $\boldsymbol{H'}$ but cannot infer $\boldsymbol{H'}$ from $\boldsymbol{H}$. Still, in a situation where the prediction x differs from the observation (or target) $\boldsymbol{x}^*$, we might want to adjust the velocity of the hidden state in the same direction than the adjustment of the hidden state. On the other hand, the inference from $\boldsymbol{H'}$ to $\boldsymbol{H}$, that we obtain according to the FEP generative model, does not seem intuitive and experimentally does not really impact the ability of the model to encode a large number of trajectories.

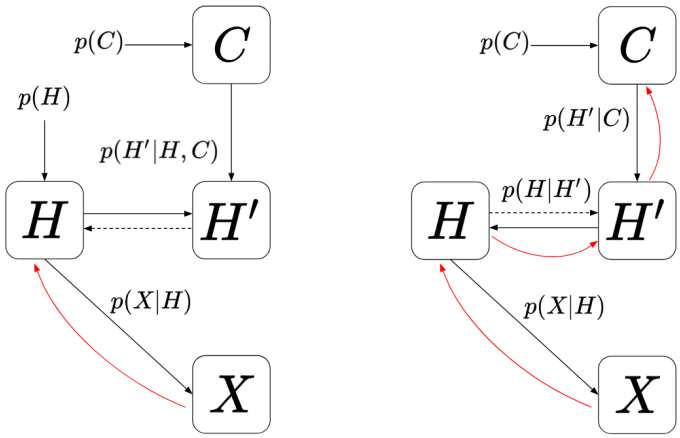


Figure 3.15: Different probabilistic models with generalized coordinates. The black arrows represent the probabilistic dependencies. The dashed arrows represent the dependencies that are ignored for inference through free-energy minimization. The red arrows represent the inference process based on the output prediction error.

For these reasons, in this section, we instead consider the probabilistic model represented on the right side of figure 3.15. In this model, we have hidden the dependency from $\boldsymbol{H}$ to $\boldsymbol{H'}$, and instead kept the dependency from $\boldsymbol{H'}$ to $\boldsymbol{H}$. The graph represented on the left corresponds to the probabilistic model typically prescribed by the FEP, see for instance (Buckley et al., 2017). With this model, there is no inference from $\boldsymbol{H}$ to $\boldsymbol{H'}$, which makes it difficult to adapt $\boldsymbol{H'}$ and in turn $\boldsymbol{C}$ based on a prediction error on the output layer.

Our probabilistic model is described by the following set of equations:

$$p(\boldsymbol{c}) = \mathcal{N}(\boldsymbol{c}; \boldsymbol{\mu}_c, \sigma_c^2 \mathbb{I}_{d_c}) \tag{3.92}$$
$$p(\boldsymbol{h}'|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{h}'; \boldsymbol{f}(\boldsymbol{c}, \boldsymbol{h}_{past}), \sigma_{h'}^2 \mathbb{I}_{d_h}) \tag{3.93}$$
$$p(\boldsymbol{h}|\boldsymbol{h}') = \mathcal{N}(\boldsymbol{h}; h' + \boldsymbol{h}_{past}, \sigma_h^2 \mathbb{I}_{d_h}) \tag{3.94}$$
$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(x; \boldsymbol{g}(\boldsymbol{h}), \sigma_x^2 \mathbb{I}_{d_x}) \tag{3.95}$$

We introduce the recognition density q:

$$q(\boldsymbol{h}, \boldsymbol{h}', \boldsymbol{c}) = q(\boldsymbol{h})q(\boldsymbol{h}')q(\boldsymbol{c}) \tag{3.96}$$
$$q(\boldsymbol{h}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{m}_h, v_h \mathbb{I}_{d_h}) \tag{3.97}$$
$$q(\boldsymbol{h}') = \mathcal{N}(\boldsymbol{h}'; \boldsymbol{m}_{h'}, v_{h'} \mathbb{I}_{d_h}) \tag{3.98}$$
$$q(\boldsymbol{c}) = \mathcal{N}(\boldsymbol{c}; \boldsymbol{m}_c, v_c \mathbb{I}_{d_c}) \tag{3.99}$$

All variables are assumed to be independent according to the recognition density, which thus factors into three marginal density functions $q(\boldsymbol{H})$, $q(\boldsymbol{H}')$ and $q(\boldsymbol{C})$. These marginal densities are assumed to be Gaussian with respective means $\boldsymbol{m}_h$, $\boldsymbol{m}_{h'}$ and $\boldsymbol{m}_c$, and respective variances v_h, $v_{h'}$ and v_c. We can write the VFE as:

$$F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c) = E(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c) + C \tag{3.100}$$

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c) = &\frac{1}{2\sigma_x^2}\|\boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)\|_2^2 + \frac{d_x}{2}\log(\sigma_x^2) \\ &+ \frac{1}{2\sigma_h^2}\|\boldsymbol{m}_h - (\boldsymbol{m}_{h'} + \boldsymbol{h}_{past})\|_2^2 + \frac{d_h}{2}\log(\sigma_h^2) \\ &+ \frac{1}{2\sigma_{h'}^2}\|\boldsymbol{m}_{h'} - \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})\|_2^2 + \frac{d_h}{2}\log(\sigma_{h'}^2) \\ &+ \frac{1}{2\sigma_c^2}\|\boldsymbol{m}_c - \boldsymbol{\mu}_c\|_2^2 + \frac{d_c}{2}\log(\sigma_c^2) \\ &+ C \end{aligned} \tag{3.101}$$

We can derive from this expression the gradient of the VFE with regard to the recognition density parameters $\boldsymbol{m}_h$, $\boldsymbol{m}'_h$ and $\boldsymbol{m}_c$:

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c)}{\partial m_{h,i}} = &-\frac{1}{\sigma_x^2}\sum_{j=1}^{d_x}\big(x_j^* - g_j(\boldsymbol{m}_h)\big)\frac{\partial g_j(\boldsymbol{m}_h)}{\partial m_{h,i}} \\ &+ \frac{1}{\sigma_h^2}\big(m_{h,i} - (m_{h',i} + \boldsymbol{h}_{past,i})\big) \end{aligned} \tag{3.102}$$

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c)}{\partial m_{h',i}} = &-\frac{1}{\sigma_h^2}\big(m_{h,i} - (m_{h',i} + \boldsymbol{h}_{past,i})\big) \\ &+ \frac{1}{\sigma_{h'}^2}\big(m_{h',i} - f_i(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big) \end{aligned} \tag{3.103}$$

$$\begin{aligned} \frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_{h'}, \boldsymbol{m}_c)}{\partial m_{c,i}} = &-\frac{1}{\sigma_{h'}^2}\sum_{j=1}^{d_h}\big(m_{h',j} - f_j(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big)\frac{\partial f_j(\boldsymbol{m}_c, \boldsymbol{h}_{past})}{\partial m_{c,i}} \\ &+ \frac{1}{\sigma_c^2}(m_{c,i} - \mu_{c,i}) \end{aligned} \tag{3.104}$$

Again, we do not consider any prior distribution on $\boldsymbol{C}$, which amounts to having a very large σ_c. We can thus remove the last term in the gradient computation with regard to $\boldsymbol{m}_c$. Reusing

the previous definition of $\boldsymbol{g}$, we obtain the following update rules for $\boldsymbol{m}_h$, $\boldsymbol{m}_{h'}$ and $\boldsymbol{m}_c$:

$$\boldsymbol{m}_h \leftarrow \boldsymbol{m}_h \underbrace{+\frac{\alpha}{\sigma_x^2}\big(1-\tanh^2(\boldsymbol{m}_h)\big)\odot\big(\boldsymbol{W}_o^\intercal\boldsymbol{\epsilon}_x\big)}_{\text{Bottom-up}} \underbrace{+\frac{\alpha}{\sigma_h^2}\boldsymbol{m}_{h'}}_{\text{Top-down}} \tag{3.105}$$

$$\boldsymbol{m}_{h'} \leftarrow \boldsymbol{m}_{h'} \underbrace{+\frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_h}_{\text{Bottom-up}} \underbrace{-\frac{\alpha}{\sigma_{h'}^2}\big(\boldsymbol{m}_{h'}-\boldsymbol{f}(\boldsymbol{m}_c,\boldsymbol{m}_h)\big)}_{\text{Top-down}} \tag{3.106}$$

$$\boldsymbol{m}_c \leftarrow \boldsymbol{m}_c \underbrace{+\frac{\alpha}{\sigma_h'^2}\boldsymbol{\epsilon}_{h'}\cdot\nabla_{\boldsymbol{m}_c}\boldsymbol{f}(\boldsymbol{m}_c,\boldsymbol{m}_h)}_{\text{Bottom-up}} \tag{3.107}$$

Similarly to the last proposed model, the function $\boldsymbol{f}$ can be implemented in an additive or in a multiplicative fashion:

$$f_{add}(\boldsymbol{m}_c,\boldsymbol{m}_h)=\frac{1}{\tau}\big(\boldsymbol{W}_r\cdot\tanh(\boldsymbol{m}_h)+\boldsymbol{W}_c\cdot\boldsymbol{m}_c-\boldsymbol{m}_h\big) \tag{3.108}$$

$$f_{mult}(\boldsymbol{m}_c,\boldsymbol{m}_h)=\frac{1}{\tau}\big((\boldsymbol{W}_R\cdot\boldsymbol{m}_c)\cdot\tanh(\boldsymbol{m}_h)-\boldsymbol{m}_h\big) \tag{3.109}$$

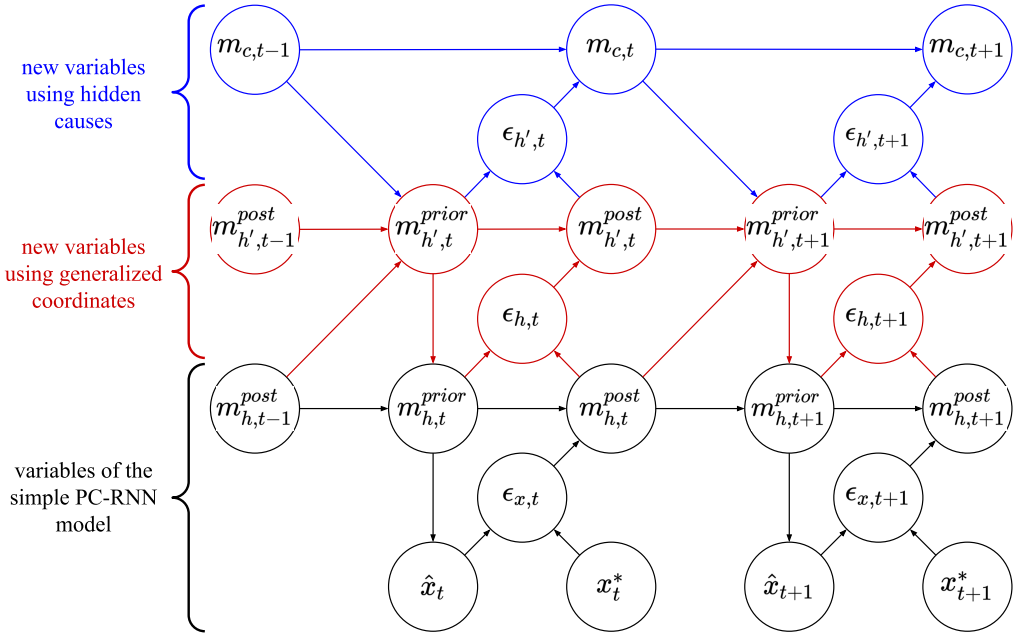


Figure 3.16: PC-RNN with hidden causes and generalized coordinates computational graph. We represent in red the variables added by the use of generalized coordinates, and in blue the variables added by the use of hidden causes.

We apply to the tensor $\boldsymbol{W}_R$ the same factorization that we have introduced previously. Both models share similar computational graphs, that can be represented as the graph in figure 3.16. The additive model is described by the following set of equations:

$$\boldsymbol{m}_{h',t}^{prior} = (1 - \frac{\alpha}{\sigma_{h'}^2})\boldsymbol{m}_{h',t-1}^{post} + \frac{\alpha}{\tau\sigma_{h'}^2}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post}) + \boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1} - \boldsymbol{m}_{h,t-1}^{post}\big) \tag{3.110}$$

$$\boldsymbol{m}_{h,t}^{prior} = \boldsymbol{m}_{h,t-1}^{post} + \boldsymbol{m}_{h',t}^{prior} \tag{3.111}$$

$$\hat{\boldsymbol{x}}_t = \boldsymbol{W}_o \cdot \tanh(\boldsymbol{m}_{h,t}^{prior}) \tag{3.112}$$

$$\boldsymbol{\epsilon}_x = \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t \tag{3.113}$$

$$\boldsymbol{m}_{h,t}^{post} = \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big) \tag{3.114}$$

$$\boldsymbol{\epsilon}_{h,t} = \boldsymbol{m}_{h,t}^{post} - \boldsymbol{m}_{h,t}^{prior} \tag{3.115}$$

$$\boldsymbol{m}_{h',t}^{post} = \boldsymbol{m}_{h',t}^{prior} + \frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_{h,t} \tag{3.116}$$

$$\boldsymbol{\epsilon}_{h',t} = \boldsymbol{m}_{h',t}^{post} - \boldsymbol{m}_{h',t}^{prior} \tag{3.117}$$

$$\boldsymbol{m}_{c,t} = \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_{h'}^2}\boldsymbol{W}_c^\intercal \cdot \boldsymbol{\epsilon}_{h',t} \tag{3.118}$$

The multiplicative hidden causes model is described by the same equations, except for equations 3.110 and 3.118 that are respectively replaced by:

$$\boldsymbol{m}_{h',t}^{prior} = (1 - \frac{\alpha}{\sigma_{h'}^2})\boldsymbol{m}_{h',t-1}^{post} + \frac{\alpha}{\tau\sigma_{h'}^2}\boldsymbol{W}_f^\intercal \cdot \Big(\big(\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post})\big) \odot (\boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1}) - \boldsymbol{m}_{h,t-1}^{post}\Big) \tag{3.119}$$

$$\boldsymbol{m}_{c,t} = \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_{h'}^2}\Big(\big(\boldsymbol{W}_p \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post})\big) \odot \boldsymbol{W}_c\Big)^\intercal \cdot \boldsymbol{W}_f \cdot \boldsymbol{\epsilon}_{h',t} \tag{3.120}$$

The pseudo-code performing a forward pass in the PC-RNN model with additive hidden causes and generalized coordinates is given in algorithm 5. It can be easily adapted for the multiplicative case.

Algorithm 5: Forward pass through the PC-RNN model with additive hidden causes and generalized coordinates.

Parameters: $\boldsymbol{W}_o, \boldsymbol{W}_r, \boldsymbol{W}_c, T, \tau, \sigma_x^2, \sigma_h^2, \alpha$
Inputs: $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$, $\boldsymbol{m}_{h,0}^{post}$, $\boldsymbol{m}_{c,0}$
for $1 \leq t \leq T$ **do**
- $\boldsymbol{m}_{h',t}^{prior} \leftarrow (1 - \frac{\alpha}{\sigma_{h'}^2})\boldsymbol{m}_{h',t-1}^{post} + \frac{\alpha}{\tau\sigma_{h'}^2}\big(\boldsymbol{W}_r \cdot \tanh(\boldsymbol{m}_{h,t-1}^{post}) + \boldsymbol{W}_c \cdot \boldsymbol{m}_{c,t-1} - \boldsymbol{m}_{h,t-1}^{post}\big)$;
- $\boldsymbol{m}_{h,t}^{prior} \leftarrow \boldsymbol{m}_{h,t-1}^{post} + \boldsymbol{m}_{h',t}^{prior}$;
- $\hat{\boldsymbol{x}}_t \leftarrow \boldsymbol{W}_o \cdot \tanh\big(\boldsymbol{m}_{h,t}^{prior}\big)$;
- $\boldsymbol{\epsilon}_{x,t} \leftarrow \boldsymbol{x}_t^* - \hat{\boldsymbol{x}}_t$;
- $\boldsymbol{m}_{h,t}^{post} \leftarrow \boldsymbol{m}_{h,t}^{prior} + \frac{\alpha}{\sigma_x^2}\big(1 - \tanh^2(\boldsymbol{m}_{h,t}^{prior})\big) \odot \big(\boldsymbol{W}_o^\intercal \cdot \boldsymbol{\epsilon}_{x,t}\big)$;
- $\boldsymbol{\epsilon}_{h,t} \leftarrow \boldsymbol{m}_{h,t}^{post} - \boldsymbol{m}_{h,t}^{prior}$;
- $\boldsymbol{m}_{h',t}^{post} \leftarrow \boldsymbol{m}_{h',t}^{prior} + \frac{\alpha}{\sigma_h^2}\boldsymbol{\epsilon}_{h,t}$;
- $\boldsymbol{\epsilon}_{h',t} \leftarrow \boldsymbol{m}_{h',t}^{post} - \boldsymbol{m}_{h',t}^{prior}$;
- $\boldsymbol{m}_{c,t} \leftarrow \boldsymbol{m}_{c,t-1} + \frac{\alpha}{\tau\sigma_{h'}^2}\boldsymbol{W}_c^\intercal \cdot \boldsymbol{\epsilon}_{h',t}$;

end

To derive this model we have modified the usual generative model considered in the FEP literature where variables representing the higher-order of motions depend on the variables representing the lower-order of motions. This change allows an inference process from the output layer towards $\boldsymbol{H}'$ and $\boldsymbol{C}$, which would not be possible otherwise. The computational model seems more intuitive, with the different layers interacting through top-down prediction, and bottom-up inference connections. We can note that in the FEP literature, the model also predicts the velocity of the observed variable, and the velocity prediction error can directly be used to infer $\boldsymbol{H}'$. Consequently, the inference problem we have raised in this section is not per se an issue brought by the FEP. It

only appears here because we disregarded the possibility of predicting the velocity of the observed variable in our probabilistic model.

Still, the modification we have brought consists in ignoring the dependency from $\boldsymbol{H}$ to $\boldsymbol{H}'$ and instead considering the dependency from $\boldsymbol{H}'$ to $\boldsymbol{H}$ in the probabilistic model. We could also imagine a model where both dependencies are taken into account, and consequently, the two inference mechanisms from $\boldsymbol{H}'$ to $\boldsymbol{H}$, and from $\boldsymbol{H}$ to $\boldsymbol{H}'$ are implemented. However, since our experiments suggest that the inference process from $\boldsymbol{H}'$ to $\boldsymbol{H}$ does not impact much the model, we have not investigated further in this direction.

About the depth of the order of motions that we model, we also limited ourselves to exploring only models with velocity $\boldsymbol{H}'$. The model we present here could be modified to account for acceleration $\boldsymbol{H}''$, jerk $\boldsymbol{H}^{(3)}$, etc.

3.3.5 Summary of the proposed models

Here we present a short summary of the different models we have derived from the FEP.

The first proposed model only comprises a hidden state variable and an output variable, it is capable of performing inference of the hidden state based on a prediction error on the output level. We label this model PC-RNN-V for Vanilla Predictive Coding-based RNN.

The second model incorporates an additional variables which is the velocity of the hidden states. It is still capable of performing inference of the hidden state based on the prediction error on the output level, but cannot infer the velocity of the hidden state. We label this model PC-RNN-GC for Predictive Coding-based RNN with Generalized Coordinates.

The third and fourth models incorporate hidden causes as a variable influencing the dynamics of the hidden states. They are capable of propagating prediction error to infer both hidden states and hidden causes. We label these two models PC-RNN-HC-A and PC-RNN-HC-M for Predictive Coding-based RNN with Hidden Causes, Additive or Multiplicative.

Finally, the fifth and sixth models use both first-order generalized coordinates and hidden causes. They are capable of propagating prediction error to infer hidden states, their velocity, as well as hidden causes. We label these two models PC-RNN-GC-HC-A and PC-RNN-GC-HC-M For Predictive Coding-based RNN with Generalized Coordinates and Hidden Causes, Additive or Multiplicative.

We summarize some of the properties of these models in the following table:

Model	Generalized coordinates	Hidden causes	Inferable variables	Hyperparameters
PC-RNN-V	✗	✗	$\boldsymbol{H}$	τ_h, α_x
PC-RNN-GC	✓	✗	$\boldsymbol{H}$	$\tau_{h'}, \alpha_x, \alpha_{h'}$
PC-RNN-HC-A/M	✗	✓	$\boldsymbol{H}, C$	$\tau_h, \alpha_x, \alpha_h$
PC-RNN-GC-HC-A/M	✓	✓	$\boldsymbol{H}, \boldsymbol{H}', \boldsymbol{C}$	$\tau_{h'}, \alpha_x, \alpha_h, \alpha_{h'}$

Table 3.2: Summary of the proposed RNN models inspired by PC.

The hyperparameters $\tau_h, \tau_{h'}, \alpha_x, \alpha_h, \alpha_{h'}$ are all obtained from the variables $\tau, \alpha, \sigma_x, \sigma_h, \sigma_{h'}$ as:

$$\tau_h = \frac{\tau\sigma_h^2}{\alpha} \tag{3.121}$$

$$\tau_{h'} = \frac{\tau\sigma_{h'}^2}{\alpha} \tag{3.122}$$

$$\alpha_x = \frac{\alpha}{\sigma_x^2} \tag{3.123}$$

$$\alpha_h = \frac{\alpha}{\tau\sigma_h^2} \tag{3.124}$$

$$\alpha_{h'} = \frac{\alpha}{\tau\sigma_{h'}^2} \tag{3.125}$$

These parameters are interdependent, but only because we have assumed that the rate α should

be the same for all the updates. If we relax this assumption, we obtain parameters that can be optimized independently.

In the last section we classified RNN models for variational inference according to several criteria. All the PC-RNN models derived in this section can perform online inference based on a prediction error signal, using a bottom-up pathway that is part of the neural architecture. Additionally, they either can be trained with BPTT, or with an online learning algorithm that does not require knowledge of future observations.

3.3.6 Possible extensions

3.3.6.1 Estimation of generation density precision

So far we have discussed the optimization of the VFE with regard to the recognition density parameters ($\boldsymbol{m}_h$, etc), and with regard to the generative model parameters ($\boldsymbol{W}_o$, etc). The first minimization process we call inference, it corresponds to the estimation of the hidden variables of the probabilistic model. The second minimization process we call learning, it corresponds to the estimation of the generative model parameters, and is suggested to happen at a slower pace. Another possibility is to optimize the generation density precision parameters:

$$p_{x,i} = \frac{1}{\sigma_{x,i}^2} \tag{3.126}$$

$$p_{h,i} = \frac{1}{\sigma_{h,i}^2} \tag{3.127}$$

In these equations, we have relaxed the assumption that the covariance matrices of the generative model were proportional to the identity matrix. Instead, we assume that they are diagonal matrices filled with the coefficients $(\sigma_{x,1}^2, \dots, \sigma_{x,d_x}^2)$ and $(\sigma_{h,1}^2, \dots, \sigma_{h,d_h}^2)$. Since the multiplication operation is more likely to be implemented by brain structures using synaptic gain, it has been porposed to use precisions instead of variances.

In (Friston and Stephan, 2007), it is suggested that learning of the model parameters and precisions operate at a slower timescale. This results, according to (Buckley et al., 2017), in an optimization of the time-integral of the free-energy by these parameters. This amounts to having second-order dynamics of the model parameters and precisions align with the gradient of the free-energy (instead of the first-order dynamics).

The simpler strategy, which we have followed when deriving and experimenting with the learning rules of the model parameters $\boldsymbol{W}_r$ and $\boldsymbol{W}_o$ in section 3.3.1, is to directly optimize these quantities with regard to the VFE (and not its time-integral) using a smaller learning rate. Here, we derive this gradient descent rule for the precision coefficients p_x, but it can be adapted to the other precision coefficients. We start by rewriting the VFE corresponding to our first proposed model with the precision coefficients:

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h) = &\frac{1}{2} \sum_{i=1}^{d_x} \Big(p_{x,i} \big(x_i^* - g_i(\boldsymbol{m}_h)\big)^2 - \log(p_{x,i}) \Big) \\ &+ \frac{1}{2} \sum_{i=1}^{d_h} \Big(p_{h,i} \big(m_{h,i} - \mu_{h,i}\big)^2 - \log(p_{h,i}) \Big) \\ &+ C \end{aligned} \tag{3.128}$$

We can derive from this expression the gradient of the VFE with regard to the precision coefficients $p_{x,i}$:

$$\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h)}{\partial p_{x,i}} = \frac{1}{2} \big(x_i^* - g_i(\boldsymbol{m}_h)\big)^2 - \frac{1}{2p_{x,i}} \tag{3.129}$$

Which gives us the following update rule for $p_{x,i}$:

$$p_{x,i} \leftarrow p_{x,i} - \frac{\lambda}{2} \big(x_i^* - g_i(\boldsymbol{m}_h)\big)^2 + \frac{\lambda}{2p_{x,i}} \tag{3.130}$$

where λ is the learning rate. To ensure that this minimization process is slower than the inference process, we typically have a value of λ that is several orders of magnitude smaller than the update rate α.

The interpretation of this update rule is surprisingly intuitive. In the presence of prediction error, the first term of this equation decreases the precision of the prediction. In the absence of error, all that remains is the second term, that increases the precision at a logarithmic pace.

We have seen in this section that the bottom-up update rules are weighted by the precision coefficients. If we were to rewrite this update rule using the vector p of precision coefficients, we would obtain:

$$\boldsymbol{m}_{h,t}^{post} = \boldsymbol{m}_{h,t}^{prior} + \alpha \boldsymbol{W}_o^{\intercal} \cdot (\boldsymbol{p} \odot \boldsymbol{\epsilon}_x) \tag{3.131}$$

We can see from this equation that features of $\boldsymbol{X}$ associated with low precisions have a reduced importance in the inference of $\boldsymbol{H}$. Consequently, modeling precisions is a very interesting prospect, as it can act as an attention mechanism on the different input features. Unfortunately, we have not experimented with this possibility during the thesis.

3.3.6.2 Stacking recurrent layers

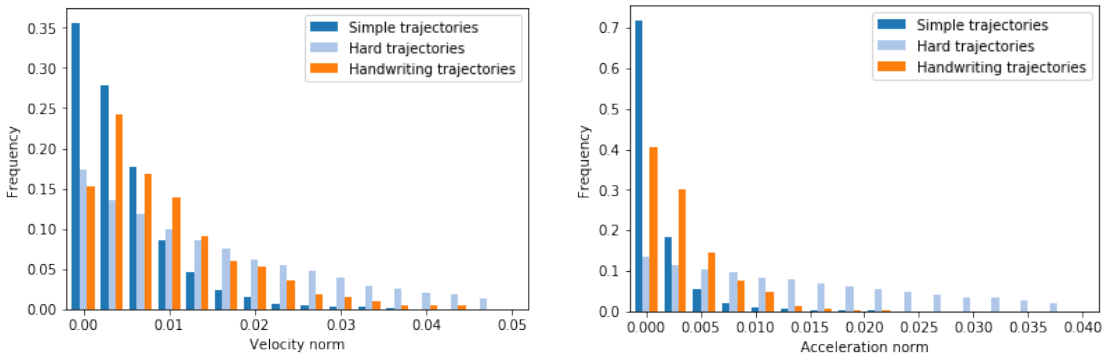
Another natural extension to our models would be to stack several layers in order to have a hierarchical generative model. This can be implemented with all the presented models, for instance using the output of the layer l as the hidden causes of layer $l-1$ in the models incorporating hidden causes.

Stacking recurrent layers might improve the generative capacity of the models, but we have not experimented with this possibility during the thesis.

3.4 Results

3.4.1 Data sets

For the experiments performed in this thesis we have used three data sets of trajectories of length 60 and dimension 2. We built the two first data sets by recording the pen position on a tablet while scribbling trajectories of variable complexity. Both data sets contain 250 trajectories. We differentiate the two data sets by the complexity of the trajectories. To increase the variability of the trajectories that were drawn, we applied random translations, rotations, shearing, and reflections onto each trajectory. This results in two data sets of random scribbles, one comprising relatively simple trajectories, while the other comprises more complex trajectories. Example trajectories from these data sets are shown in figure 3.18. The sorting of the trajectories between the simple and hard data sets was made subjectively. To validate that our sorting was efficient, we analyze the distributions of velocity and acceleration within each data set.



(a) Distribution of the velocity norm.

(b) Distribution of the acceleration norm.

Figure 3.17: Data sets statistics.

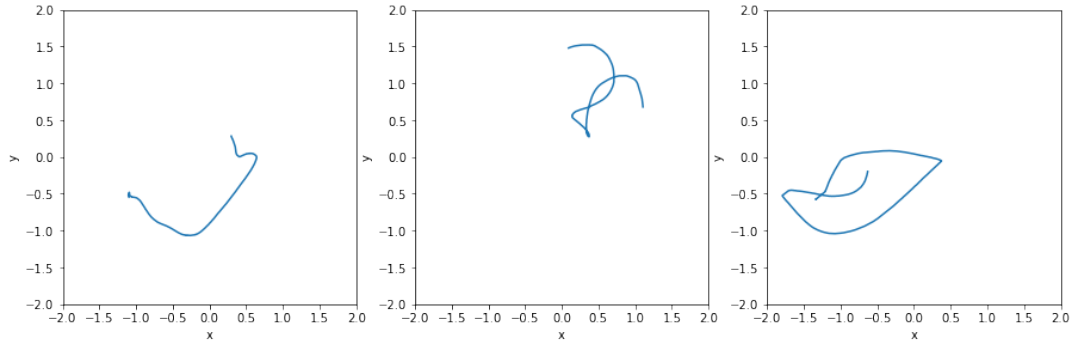
The last data set (Dua and Graff, 2019) is composed of trajectories corresponding to recorded (x, y) positions of human letter handwriting. The data set comprises around 40 trajectories for

each letter of the latin alphabet not requiring to lift the pen. We also analyze its characteristics to evaluate whether its trajectories are overall simpler or more complex than the ones in the two other data sets.

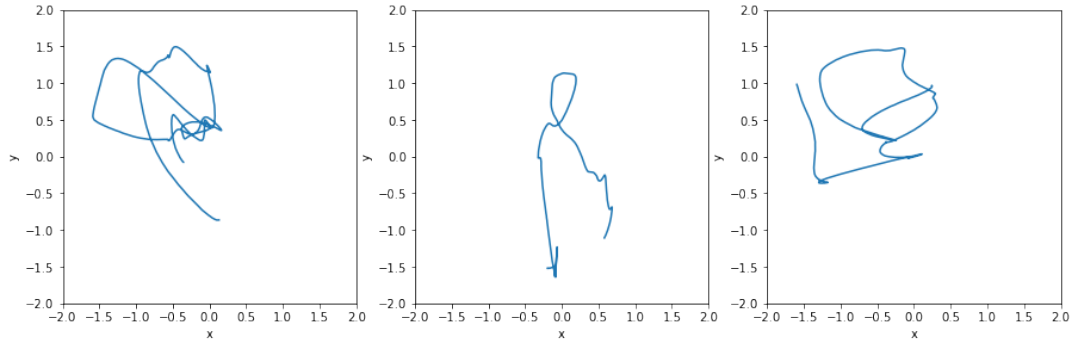
Characteristics of the three data sets are represented in figure 3.17. The distribution of velocity and acceleration confirm that overall the hard data set is composed of trajectories with more variations of velocity and acceleration than the simple data set. The handwriting data set appears to lay in between the two others in terms of complexity.

Some works in the literature aim at quantifying more precisely the complexity of trajectories. For instance, (Hug et al., 2021) suggests aligning all trajectories before performing a clustering on the data set. The data set complexity is evaluated as a combination of the number of clusters, the intra-cluster variance, and the inter-cluster variance.

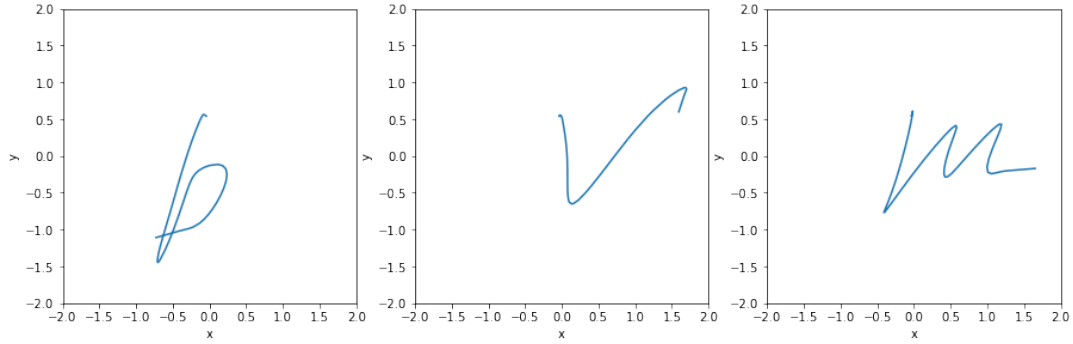
Some examples of trajectories of the three data sets are represented in figure 3.18.



(a) Example trajectories from the simple data set.



(b) Example trajectories from the hard data set.



(c) Example trajectories from the handwriting data set.

Figure 3.18: Example trajectories from the three data sets.

3.4.2 Learning, prediction and inference

To simplify discussions, we introduce three configurations for the presented models, that we call prediction, inference and learning modes.

3.4.2.1 Prediction

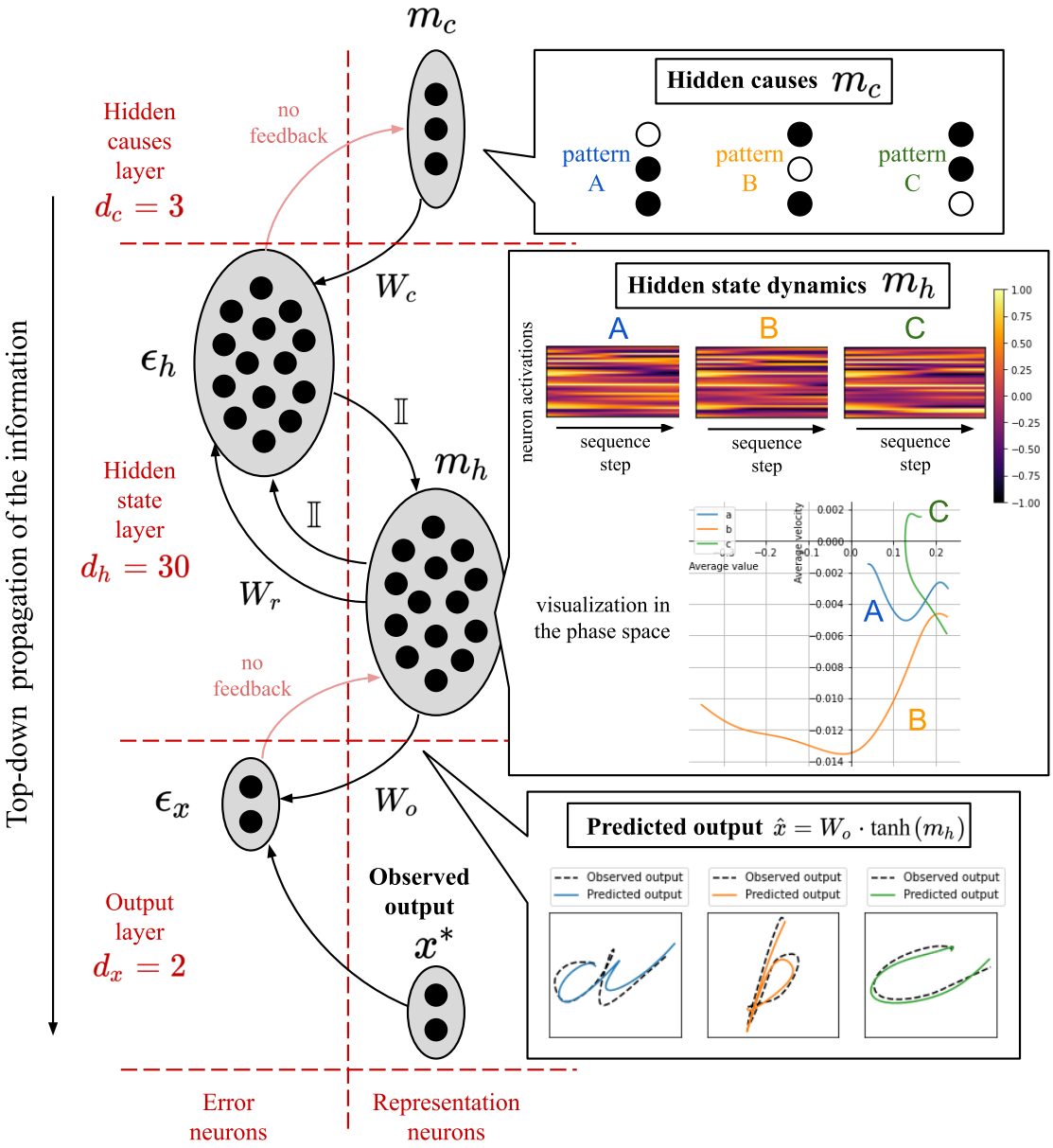


Figure 3.19: Illustration of the PC-RNN-HC-A prediction on a toy example.

In *prediction* mode, only the generative model part of the RNN is active. This mode does not require a target output sequence $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$, and simply computes the prediction based on the provided input. This mode is relevant when evaluating our models as sequence memories, and when comparing their prediction accuracy with other models not incorporating feedback connections. To turn off all the bottom-up computations, we set to 0 the values of the update rates $\alpha_x, \alpha_h, \alpha_{h'}$. No learning is performed in prediction mode.

To illustrate this prediction process, we represent in figure 3.19 the evolution of the variables encoded in the different neural network layers during prediction. We use a PC-RNN-HC-A model trained to generate three sequence patterns corresponding to handwritten letters. We use a hidden state dimension $d_h = 30$ and hidden causes dimension $d_c = 3$, the time constant of the RNN is set to $\tau_h = 30$. After training, we can see that using the three possible one-hot inputs for the hidden causes yields predictions corresponding to the three patterns a, b and c. The figure displays the evolution of the hidden state activation through this prediction process, as raster plots and as trajectories in phase space (right). In phase space, we represent the hidden state trajectory using the average layer activation as horizontal coordinate and the average layer activation velocity as

vertical coordinate. We observe that with three different hidden causes initializations, the hidden state dynamics are different even in the absence of feedback. Finally, we display on the bottom the three predicted output trajectories, as well as the observed trajectory in dashed black lines.

The source code used to train the PC-RNN-HC-A model and to generate the figures 3.19 and 3.20 is available on GitHub[1].

3.4.2.2 Inference

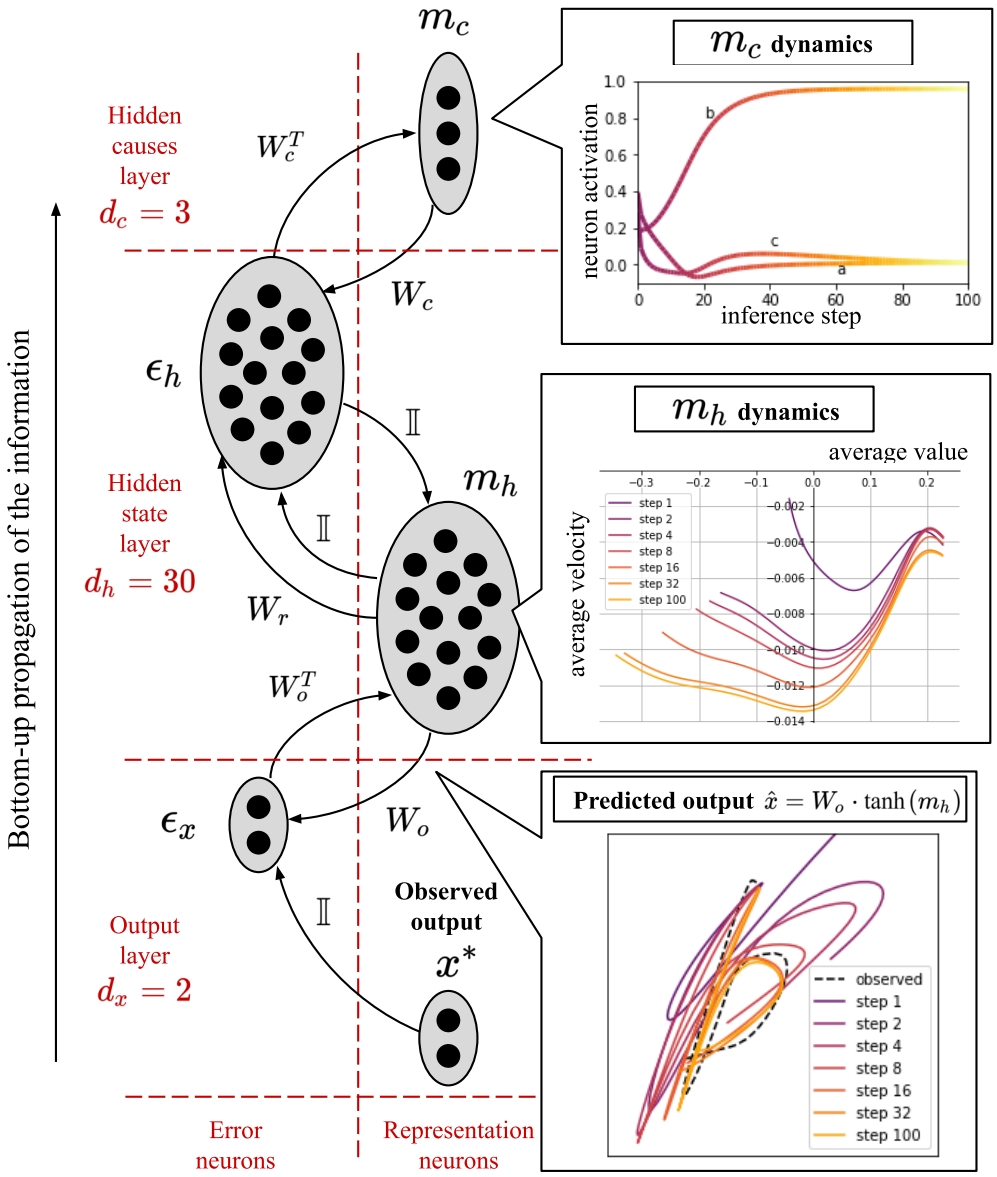


Figure 3.20: Illustration of the PC-RNN-HC-A inference algorithm on a toy example.

In *inference* mode, we authorize the dynamic update of the latent variable of the models based on a target output sequence $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$. Thus, the values of the update rates $\alpha_x, \alpha_h, \alpha_{h'}$ are kept positive.

In models incorporating hidden causes, the inference process also updates the estimate of these quantities. Crucially, the initial hidden causes *are* the input of those models, which makes it possible to infer the input based on the desired output (the target sequences). This inference mechanisms can thus approximate an inverse of the function $\boldsymbol{c} \in \mathcal{C} \rightarrow (\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_T) \in \mathcal{X}^T$ implemented by our sequence memory model. This feature is of particular interest when we

[1] https://github.com/sino7/example_pc_rnn

consider the questions of continual learning and memory retrieval. As in the *prediction* mode, no learning is performed in *inference* mode, only the latent variables of the model are updated.

To illustrate this inference process, we display in figure 3.20 the evolution of the hidden causes, the hidden states dynamics induced by these hidden causes, and the resulting predicted output trajectories, through 100 inference steps using a pre-trained PC-RNN-HC-A model. As before, the PC-RNN-HC-A model has been trained to generate three sequence patterns corresponding to the handwritten letters *a*, *b* and *c*. We use a hidden state dimension $d_h = 30$ and hidden causes dimension $d_c = 3$, the time constant of the RNN is set to $\tau_h = 30$. In this experiment, we turn on the feedback pathway with $\alpha_x = 0.002$ and $\alpha_h = 1$. The hidden causes are initialized in a neutral configuration with no prior belief about the category of the observed trajectory. The initial hidden causes are sampled from a normal distribution of mean $(1/3, 1/3, 1/3)$ and standard deviation 0.1. We provide an observed output trajectory $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$ corresponding to the pattern *b* in the data set. This trajectory is represented by black dashed lines in the bottom figure. We call an inference step the complete prediction of the output trajectory (60 time steps), with the associated bottom-up update of the hidden causes.

The top figure displays the evolution of the hidden causes through 100 inference steps. We observe that after about 50 inference steps, the hidden causes representation has converged towards a value very close to the one-hot vector activated on the second neuron, which corresponds to the hidden causes value that was associated with the pattern *b* during training. As such, the inference process has performed a classification of the observed sequence by updating the internal belief encoded in the hidden causes layer. The middle figure represents several hidden state trajectories in the phase space, through the inference process. We can see that after 32 inference steps, the hidden state dynamics are almost stable. The bottom figure represents several predicted output trajectories through the inference process. We can see that the predicted trajectories converge to a limit trajectory closely approximating the observed trajectory.

3.4.2.3 Learning

Finally, we discuss shortly the different learning methods that we have used for the proposed models. For each PC-RNN model, we can derive learning rules for synaptic weights that directly minimize VFE. Such rules can be derived as well for bias parameters, something we have not included in the equations for simplicity, and for all the models presented in the previous section.

When applying these learning rules, it is necessary to keep the coefficients α_x, α_h and $\alpha_{h'}$ non-null as they are necessary to propagate the prediction error signal in the upper layers.

The second possible learning method we consider is BPTT, that we have introduced previously in chapter 2. (Millidge et al., 2020a) reported that training neural networks using PC approximated the BP algorithm. However, learning with PC was computationally more expensive as many iterations on each data sample were necessary for the inference process to converge before applying the learning rules on the parameters.

In our case, the PC learning algorithm is slightly different from their proposition for two reasons. First, we do not authorize the inference of variables of the computational graph that belong to the past, such as the past hidden states. Second, we synchronized the inference and learning process with the temporal dynamics of the generative model, only allowing one inference iteration of the latent variables at each time step. These differences make one iteration of learning with PC faster than the version proposed in (Millidge et al., 2020a), but should also lead to worse solutions.

Experimentally, we have obtained faster convergence and better learning using the BPTT algorithm instead of these PC learning rules, without even resorting to gradient descent optimizers for BPTT. In the experiments presented in this section, we have used the PC-based learning rules.

Mode	Available target	$\alpha_x, \alpha_h, \alpha_{h'}$
Prediction	No	0
Inference	Yes	>0
Learning (PC)	Yes	>0
Learning (BPTT)	Yes	0

Table 3.3: Prediction, inference and learning modes.

We refer to *learning* modes the configurations of hyperparameters α_x, α_h and $\alpha_{h'}$ used for PC-based learning and BPTT. As already explained, during PC-based learning, these coefficients are positive. However, for BPTT, we found better convergence when setting those parameters to 0. This could come from the fact that the inference process slightly deviates the trajectory of hidden states from its natural dynamic, something that interferes with the learning process. When using the PC based learning rules, we also obtained better results with very low values of α_x, compensated by high values of λ_r and λ_c. In summary, it seems better to train the model parameters according to the usual deep learning methodology, using BPTT, and without any inference process interfering.

Table 3.3 provides the hyperparameters configurations for each of the modes we have discussed.

3.5 Conclusion

In this chapter, we have derived several RNN models from the free-energy formulation of PC. Taking inspiration from the FEP literature, we have tried using generalized coordinated and hidden causes in our RNN designs. The obtained models can be trained as sequence memories. Using these models in *prediction* mode corresponds to the reading operation in the sequence memory. Using these models in *learning* mode corresponds to the writing operation in the sequence memory. Finally, the PC inspiration makes it possible to perform *inference* of the latent causes of a given target pattern. As we will see in chapter 5, this *inference* mechanism can be used to perform content-based addressing, i.e. retrieve learned patterns from their content.

In the next chapter, we will focus on evaluating the writing mechanisms (i.e. learning) in the proposed sequence memory models.

Chapter 4

Comparative studies

4.1 Introduction

In this chapter, we present two comparative studies featuring the models derived in the previous chapter. In the first study, we evaluate our models together with many RNN implementations from the literature on the question of memory capacity. All models are trained as sequence memories using the *hard* data set introduced previously, and we estimate their capacity as the ratio between the number of learned sequential patterns and the number of model parameters. Our models are trained using the BPTT algorithm.

In the second study, we compare the PC-based learning algorithm with other learning methods that can be performed completely online, with update rules involving only local information. The models are evaluated with regard to their ability to extend to a continual learning setting. For each model we incrementally learn a repertoire of sequences from the *simple* data set, and measure the obtained prediction error at the end of training.

This chapter is thus composed of two sections. In section 4.2, we present the first comparative study, and in section 4.3 we present the second comparative study.

4.2 Model capacity

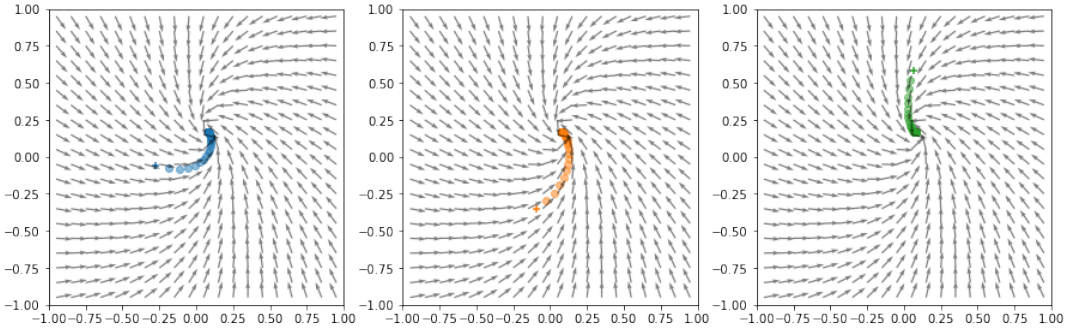
In this section, we evaluate the proposed models as well as benchmark models from the literature on the question of sequence generation capacity. We define this capacity as the ratio between the number of sequence patterns *properly learned* by the model, and the number of tunable model parameters during learning. The criterion we used to count the number of sequence patterns properly learned is completely subjective, and other methods could be more relevant. We simply defined a threshold value for the average prediction error during the trajectory. If the predicted trajectory departs from the target trajectory by a thinner margin than this threshold, then the target trajectory is considered properly learned by the model. During our experiments, this threshold value was set to 0.1. We denote n the number of target trajectories properly learned by a model.

4.2.1 Intuition

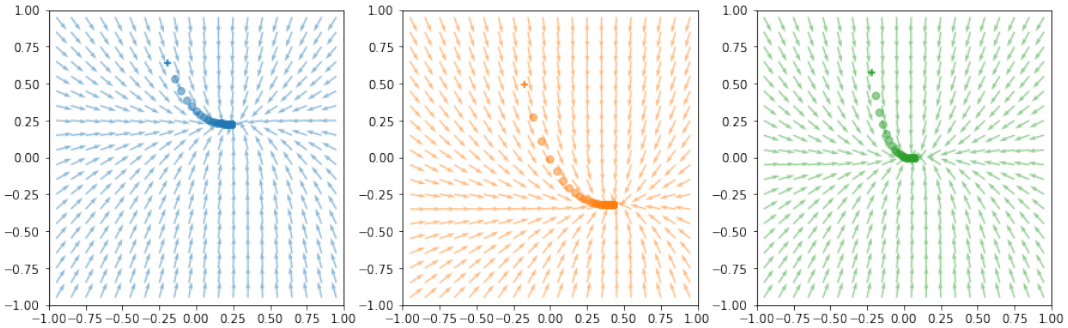
In standard RNNs, the trajectories are all generated by the same dynamical system based on different initial states. In contrast, in RNN models incorporating hidden causes, different hidden causes values define different dynamical systems as a whole. In the additive hidden causes models, different hidden causes correspond to different biases $\boldsymbol{W}_c \cdot \boldsymbol{m}_c$. In the multiplicative model, different hidden causes correspond to different recurrent weight matrices $\boldsymbol{W}_R \cdot \boldsymbol{m}_c$, where for simplicity we ignore the factorization method we introduced and $\boldsymbol{W}_R$ denotes the three-way tensor. This bias and this recurrent matrix are both involved in the equation computing of the current hidden state based on the past hidden state. Consequently, each hidden causes value defines a different dynamical system characterized by this equation.

To illustrate this idea, we have analyzed the trajectories generated by three models in a sandbag experiment where the hidden state dimension is $d_h = 2$. With only two dimensions, it is easy to visualize the temporal evolution of the hidden states. Although this evolution could be visualized

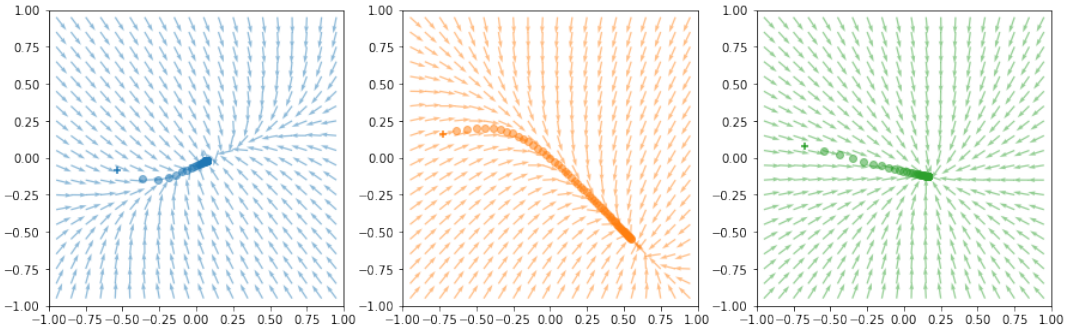
in other forms with a dimension greater than two, this also makes it possible to draw a vector field representing the dynamics of the RNN at each point of the state space.



(a) Example dynamics of the hidden states with three different initializations $\boldsymbol{h}_0$, with the PC-RNN-V model.



(b) Example dynamics of the hidden states with the same initialization $\boldsymbol{h}_0$ but three different hidden causes c_0, with the PC-RNN-HC-A model.



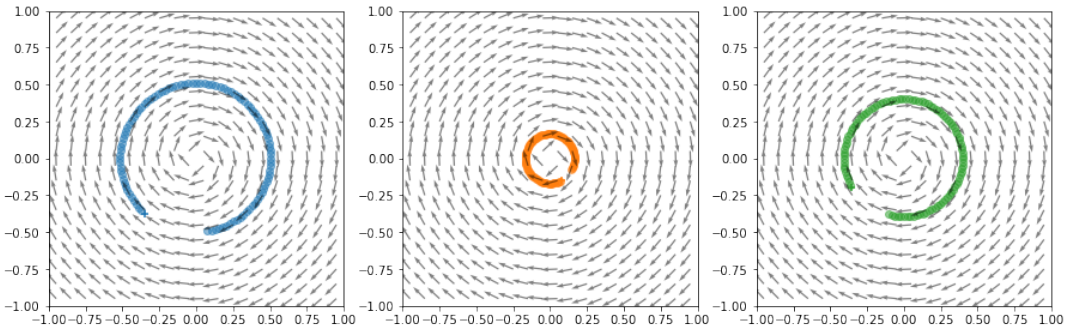
(c) Example dynamics of the hidden states with the same initialization $\boldsymbol{h}_0$ but three different hidden causes c_0, with the PC-RNN-HC-M model.

Figure 4.1: Example hidden states dynamics with three of the proposed models: PC-RNN-V, PC-RNN-HC-A and PC-RNN-HC-M.

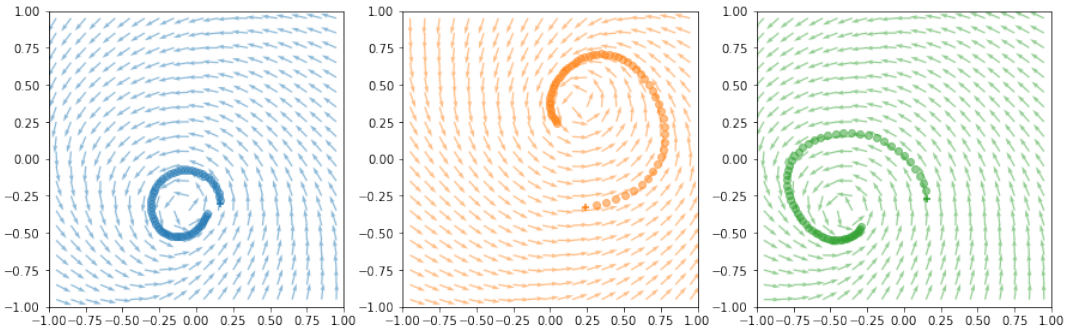
Figure 4.1 presents trajectories obtained with this idea, using three models: the PC-RNN-V, the PC-RNN-HC-A and the PC-RNN-HC-M. In the top row, we have represented three trajectories obtained with a randomly initialized PC-RNN-V model using random initial hidden states. The variability of trajectories might be limited by the fact that they are all generated by the same dynamical system. In this case, since the dynamical system comprises a point attractor, all trajectories converge to the same point.

In the middle and bottom rows, we have represented trajectories obtained with the same initial hidden state, but with different hidden causes values, for the PC-RNN-HC-A and PC-RNN-HC-M models. The three hidden causes values define three different dynamical systems, as highlighted by the different colors of the vector fields. We can also observe that the three dynamical systems may differ more in the multiplicative case than in the additive case.

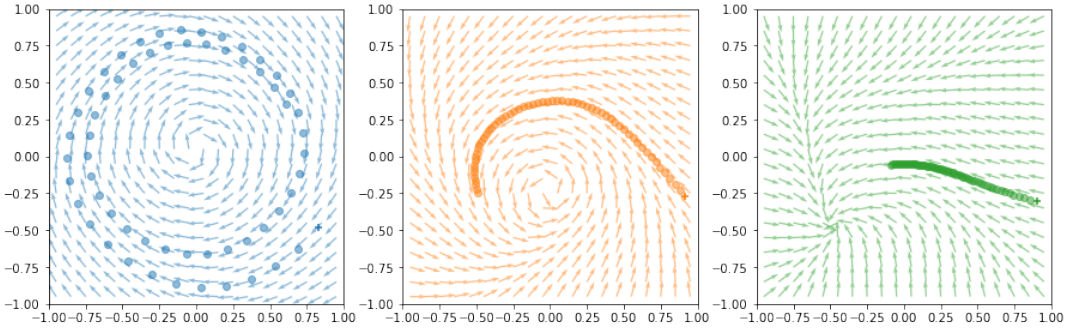
This visualization method is limited because our dynamical systems are very simple. To allow



(a) Example dynamics of the hidden states with three different initializations $\boldsymbol{h}_0$, with the PC-RNN-V model.



(b) Example dynamics of the hidden states with the same initialization $\boldsymbol{h}_0$ but three different hidden causes c_0, with the PC-RNN-HC-A model.



(c) Example dynamics of the hidden states with the same initialization $\boldsymbol{h}_0$ but three different hidden causes c_0, with the PC-RNN-HC-M model.

Figure 4.2: Example hidden states dynamics with three of the proposed models: PC-RNN-V, PC-RNN-HC-A and PC-RNN-HC-M, using an antisymetric initialization.

better visualization while remaining in dimension $d_h = 2$, we have initialized recurrent weights to be antisymmetric matrices. The obtained trajectories are represented in figure 4.2. Because of this antisymmetric initialization, our RNNs yield dynamics that do not converge directly to point attractors, as suggested in (Chang et al., 2019).

These new figures strengthen our intuition that multiplicative influence of the hidden causes onto the hidden states dynamics might induce more variability in the obtained trajectories. In the middle row, the three vector fields are very similar. In the bottom row, the first vector field shows dynamics that rotate in the clockwise direction while the second vector field shows dynamics that rotate in the counterclockwise direction. Moreover, the third vector field describes a dynamical system comprising a point attractor, while the first two vector fields describe a cycle attractor.

With this examples, we have built an intuition that generative models comprising hidden causes layers might be able to generate more variable trajectories in the hidden state space. This variability could induce a better trainability or a better capacity of those models. As we will see, the results presented in this section partially contradict this intuition.

4.2.2 Benchmark models

To evaluate our models, we perform a comparative analysis of their generative capacities with a set of benchmark RNN models previously introduced in section 3.2.

The first benchmark model that we use is the TRNN, that uses the same generative structure as the PC-RNN-V, with the difference that it does not allow inference of latent variables. To study the effects of gating mechanisms onto generative capacity, we use the LSTM (three gates), GRU (two gates) and UGRNN (one gate) models. To study the effects of hierarchical structure, we use an MTRNN model with two layers (and thus two time constants). To also cover models that incorporate constraints on the recurrent weights, we use the AntisymetricRNN. Finally, we also compare performances with an ESN, to include the RC paradigm into this study.

4.2.3 Hyperparameter optimization

For all of the models that we derived in section 3.3, and all the benchmark models listed above, we perform an optimization of hyperparameters using Gaussian processes. There are many hyperparameters in RNN models, which we could not all optimize because of time constraints, and the number of models considered in our study. Hyperparameter optimization was performed using Bayesian optimization with Gaussian processes and Matern 5/2 Kernel, similarly to the RNN encoding capacity comparative analysis performed in (Collins et al., 2017).

Basically, this method tries to approximate the function $\boldsymbol{P} \rightarrow f(\boldsymbol{P})$ that associates a scoring function defined by our hands with a certain hyperparameter configuration $\boldsymbol{P}$. This approximation is estimated based on points $\big((\boldsymbol{P}_0, f(\boldsymbol{P}_0)), (\boldsymbol{P}_1, f(\boldsymbol{P}_1)), \cdots\big)$ sampled sequentially by the optimizer. We call acquisition function the function used by the optimizer to guide its sampling process. Here, we used an expected improvement acquisition function, meaning that at each iteration, the optimizer samples the point P that is most likely to improve the current estimated maximum of the function f. Compared to exhaustive hyperparameter optimization methods such as random search or grid search, this method is expected to converge faster and to better configurations. To perform this hyperparameter optimization we used the `gp_minimize` function from the scikit-optimize library in python.

Initialization of weight and bias parameters plays an important role in the trainability of RNNs. During hyperparameter search however, we do not optimize the hyperparameters responsible for the initial parameters of our models. Such hyperparameters include the variances of the independent multivariate normal distribution from which we sample initial weights and biases, or the mean of the initial biases of gated networks, that can be initialized according to the expected time constant of the network (Tallec and Ollivier, 2018).

However, we make an exception with the ESN model. Indeed, ESNs naturally trade trainability for hyperparameter optimization. Biological inspiration as well as theoretical foundations provide us guidelines for weights initializations that yield complex non-linear self-sustained dynamics, particularly suited for our problem. This initialization makes the model easier to train.

The hyperparameters that were optimized are, depending on the models, the learning rate of the gradient descent optimizer λ, the time constants of the RNN models, the initial connectivity probability of the recurrent weights p_{init} (only for the ESN), the initial scale of the recurrent weights σ_{init} (only for the ESN), and finally, the number of target trajectories p considered for training.

Optimizing the value of p might be a bit surprising since it is a parameter controlling the difficulty of the task instead of the model or learning algorithm. However, this is a simple way to find the maximum number of sequence patterns properly learned by the model, i.e. the maximum value for n, without having to try all values of p by hand. Of course, searching the value of p for instance using dichotomy would be faster than exhaustive search, but including this search in the hyperparameter search highly decreases the time spent to find suitable hyperparameters and number of target trajectories to train on.

We deliberately chose not to include the number of training iterations in this hyperparameter optimization process, for several reasons. First, each hyperparameter to be optimized makes the hyperparameter search longer. Second, it is very probable that the hyperparameter optimizer would simply find that the best value is the upper bound of authorized values for the search. Indeed, training longer should only lead to better prediction accuracy. To ensure a fair comparison between all models trained with BPTT (i.e. all models except the ESN), we set the number of

training iterations to 3000. This limit seemed like a proper trade off between performance and time consumption, but it is completely subjective.

The update rates in PC-RNN models were also not considered, except for $\alpha_{h'}$ in the PC-RNN-GC model. As already reported when discussing learning algorithms, early results with these models indicated that turning off the inference pathways (by setting α_x to 0) improved prediction accuracy after learning. The hyperparameter $\alpha_{h'}$ is slightly different, as it controls the inference of $\boldsymbol{H}$ based on the prediction error on the layer $\boldsymbol{H'}$. This inference process was guided by the free-energy formulation of the RNN models but did not seem very intuitive. Here we allow this hyperparameter to be optimized and the optimizer found best results with minimal values of $\alpha_{h'}$. This validates our intuition, explained in section 3.3.4 that the inference of $\boldsymbol{h'}$ based on the prediction on $\boldsymbol{h}$ might be more interesting in our model derivations.

Let us now discuss the scoring function that we used. The most intuitive function would be to return the number of target trajectories that were properly learned, n, according to the criterion we explained previously. However, this scoring function might not be the best fit since it does not account for the fact that higher values of p also mean a higher number of model parameters. Indeed, all models suppose an input in the shape of a one-hot vector of dimension p. For models incorporating hidden causes, this input is used as initial hidden causes. For other models, this input is used to compute the initial hidden state $\boldsymbol{h}_0$. In both cases, the weight matrix processing this input is of width p, and consequently the total number of model parameters increases linearly with p.

The scoring function should thus include a penalty for hyperparameter configurations that increase the number of model parameters without increasing n. For instance, if a model can properly encode 10 trajectories when trained on 10 trajectories, and still 10 trajectories when trained on 20, it would have a better capacity in the first configuration since this corresponds to a smaller number of parameters. We experimented with different ways of implementing this penalty, and found most success using a scoring function returning $n - (p - n)$. Intuitively, this scoring function counts the number of trajectories properly learned, with a penalty amounting to the number of trajectories that were trained on but not properly learned.

For each model, we performed this hyperparameter optimization for 3 different values of the hidden state dimension d_h, except for the ESN model where we used 4 values of d_h (simply because training time and thus hyperparameter optimization was way faster with this model). Each optimization was limited to 300 iterations.

Model	Optimized hyperparameters
TRNN	p, λ, τ
LSTM	p, λ
GRU	p, λ
UGRNN	p, λ
AntisymmetricRNN	p, λ
MTRNN	$p, \lambda, \tau_f, \tau_s$
ESN	$p, \tau, p_{init}, \sigma_{init}$
PC-RNN-V	p, λ, τ_h
PC-RNN-GC	$p, \lambda, \tau_h, \alpha_{h'}$
PC-RNN-HC-A	p, λ, τ_h
PC-RNN-HC-M	p, λ, τ_h
PC-RNN-GC-HC-A	p, λ, τ_h
PC-RNN-GC-HC-M	p, λ, τ_h

Table 4.1: Optimized hyperparameters for each RNN model.

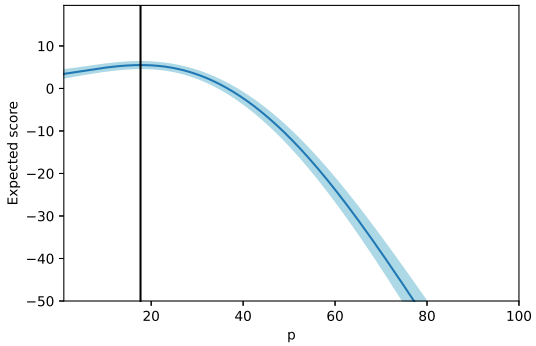
Table 4.1 provides a list of the hyperparameters optimized for each model, where τ denotes

the time constants of the ESN and TRNN, and τ_f and τ_s (for *fast* and *slow*) denote the two time constants of the MTRNN.

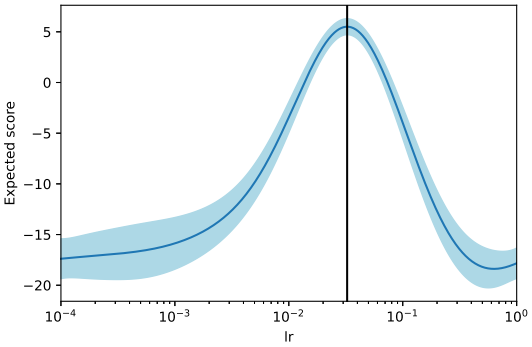
4.2.4 Comparative analysis

The source code for the experiments presented in this section is available on GitHub[1].

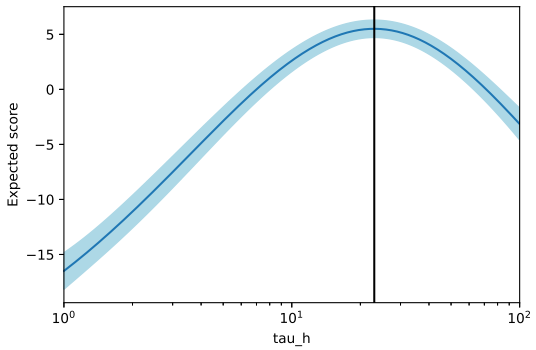
We perform the comparative analysis according to the presented experimental set up, using the *hard* data set. We choose this data set, because the handwriting data set does not comprise a large variety of sequential patterns, which we need to properly compare the models' capacity. On the other hand, the *simple* data set comprises trajectories that were very easy to learn by the models. To properly study capacity, we would have had to use very small model hidden dimensions d_h, which can lead to undesirable limit case behaviors.



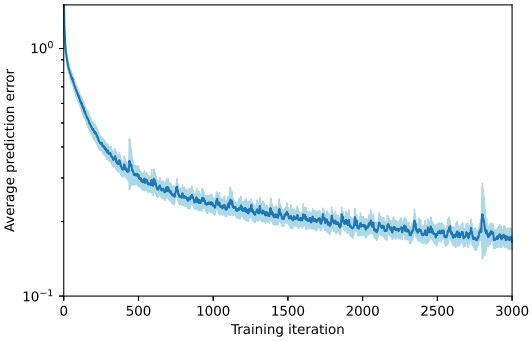
(a) Score estimation of the hyperparameter optimizer with regard to p.



(b) Score estimation of the hyperparameter optimizer with regard to the learning rate λ.



(c) Score estimation of the hyperparameter optimizer with regard to the time constant τ_h.



(d) Average prediction error over 10 seeds for the training of the model with the optimal hyperparameters.

Figure 4.3: Hyperparameter optimization and model training for the generative capacity comparative study. These results were obtained for the PC-RNN-V model with $d_h = 60$.

We first present one example of hyperparameter optimization results. Figure 4.3 displays the approximation of the scoring function estimated by the hyperparameter optimizer after 300 iterations for the PC-RNN-V model, as well as the evolution of the average prediction error with the PC-RNN-V model using the found optimal hyperparameters (figure 4.3d). After 300 iterations, the hyperparameter optimizer has built an estimate of the score function with an easily identifiable maximum. The standard deviation of this estimation is represented in the figures, and is lower near the maximum point. This is because the optimizer has sampled more points near the optimum and thus can more precisely estimate the expected scores for these points.

After identifying the optimal configuration of hyperparameters from this estimation, we use it to perform 10 seeds of training and evaluation of the model. Figure 4.3d reports the evolution of the prediction error, averaged over these 10 seeds.

[1] https://github.com/sino7/rnn_generative_capacity_benchmark

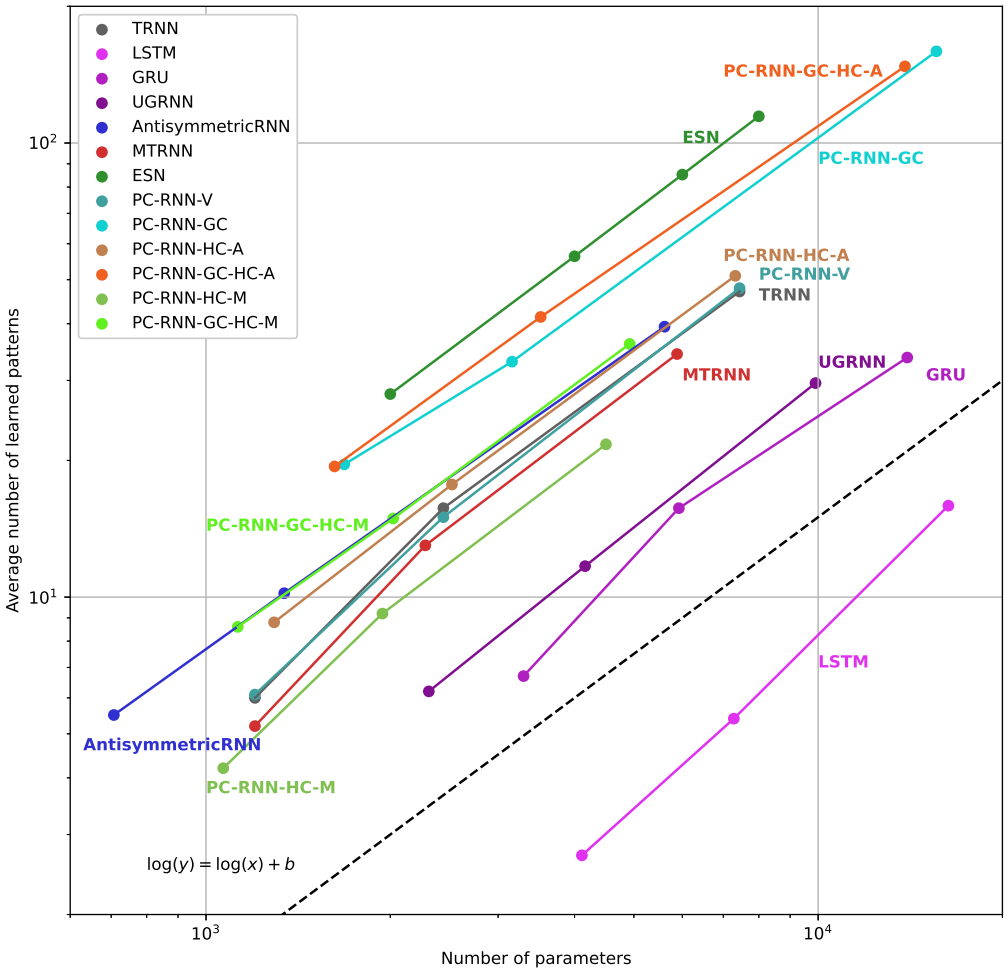


Figure 4.4: Comparative analysis of the models' generative capacity.

Finally, for each model, we measure the average number of trajectories properly learned by the model across these 10 seeds, as well as the number of trainable parameters of the models. The results are reported in figure 4.4 where we represent the expected number of learned trajectories according to the number of trainable parameters of each model.

From this figure emerges a clear relation between the number of learned trajectories and the number of parameters. In logarithmic scale, these two quantities seem to be related according to the equation $\log(y) = \log(x) + b$, with a different value of b for each model. If we remove the logarithms, we obtain that for each model, the number of learned trajectories is proportional to the number of trainable parameter.

This result confirms that we can define a notion of generative capacity as the ratio between the number of learned trajectories and the number of trainable parameters, a quantity that seems constant for different sizes of the same model. Note that if we were to compare the number of learned trajectories with the dimension of the hidden state, or with the total number of synapses, we would not obtain this clear relation for all models.

We now shortly develop the observable differences between the model capacities. In an effort to bring some nuance to the following comments, we insist on the fact that they are limited to the scope of the data set on which we performed the comparison. Additionally, the observed differences might to some extent depend on our implementation choices, parameter initialization, choice of hyperparameters to optimize, and other factors.

- **TRNN and PC-RNN-V are equivalent:** We have already established that the PC-RNN-V model was equivalent to a standard TRNN if we remove the bottom-up connections. We have also explained that we obtained better results on PC-RNN models when omitting these connections during learning, so it is not surprising that the PC-RNN-V and TRNN

model show very similar curves. This result is still interesting, as it increases our confidence in our experimental set up, since the hyperparameter optimizer found nearly the same hyperparameters for both models.

- **Gating reduces capacity:** The results seem to demonstrate that gating reduces generation capacity. This is also observed in (Collins et al., 2017) to a lesser extent, and on a different task. Within our panel of models, models that do not incorporate gating mechanism perform better than, in order, the UGRNN (one gate), the GRU (two gates), and the LSTM (three gates) models. This result might only hold for generation tasks where the data set trajectories are continuous and low-dimensional, as the gating mechanisms might be better suited to processing complex data with delayed causal relations.

- **Using multiple layers slightly decreases capacity:** We observe a slightly reduced generative capacity for the MTRNN model compared to the TRNN model. However, this comparison might only hold on this specific data set, and the difference is not significant. At first, it might seem that the TRNN model only constitutes a special case of MTRNN model where all time constants are equal. However, there still remains a difference in connection patterns as the layers of the MTRNN only interact with directly adjacent layers. We can also relate that the hyperparameter optimizer always found optimal configurations where $\tau_f < \tau_s$, confirming the intuition that MTRNN models should be initialized with longer time constants in the upper layers.

- **The antisymetric constraint on recurrent weights increases capacity:** The AntisymetricRNN slightly outperforms the TRNN. This suggests that the antisymetric constraint, allowing to decrease the number of model parameters, is an interesting feature that could be exported to other models. Still, the difference is not significant and might be due to improved trainability rather than the structural constraint on the model. Since we have limited the number of training iterations for all models, models with faster and more stable training are advantaged by our method.

- **First-order generalized coordinates improve capacity:** Within the PC-RNN models, it appears that the generalized coordinates version of a model always significantly outperforms the base model. This could come from the fact that the data set that was used to perform this comparison is composed of trajectories drawn by hand. It is possible that it is easier to generate the first-order derivatives of such trajectories. Using second-order generalized coordinates might also have improved the generative capacity, but we have not experimented with this idea.

- **Multiplicative hidden causes reduce capacity:** The models incorporating multiplicative hidden causes (PC-RNN-HC-M and PC-RNN-GC-HC-M) have a lower generative capacity than the base models (PC-RNN-V and PC-RNN-GC). This result comes as a surprise, as our intuition was that this implementation choice could improve capacity, as explained in section 4.2.1, and based on the success of the Multiplicative RNN (Sutskever et al., 2011).

- **Additive hidden causes slightly improve capacity:** On the other hand, the models incorporating additive hidden causes (PC-RNN-HC-A and PC-RNN-GC-HC-A) outperform their base models (PC-RNN-V and PC-RNN-GC). Although not significant, this improvement is interesting considering that hidden causes model also have the advantage of allowing online inference of their input, something that is not possible with PC-RNN-V and PC-RNN-GC models.

- **RC outperforms other approaches:** Finally, the ESN model outperforms all other models. This is interesting, since it adds to the advantage of a highly reduced training time. This difference could come from the fact that the optimal weights for the ESN model are found analytically whereas the weights of other models are local minima found using gradient descent. We should still note that this increased capacity comes with the downside of having to deal with very large neural networks compared to other models, with the associated memory and computation costs during generation.

4.3 Continual learning and catastrophic forgetting

We have so far presented recurrent neural architectures inspired by PC that can be used for sequence memory modeling. A key feature of these models is the corresponding computations can potentially be performed in an online fashion, each neuron activation and synaptic connection weight being updated using directly available information. However, although our models can be trained properly using a continuous stream of target values, we have always assumed that the target sequences for learning were independent and identically distributed. In this section, we study the ability of our models, as well as other methods, to be trained in a continual learning setting. This section is organized as follows. First, we expose the problem of continual learning. Then, we present our experimental set up as well as the models we have selected for comparison. Finally, we discuss our results and try to identify the online learning mechanisms that are relevant for continual learning.

Continual learning is a branch of machine learning aiming at equipping learning agents with the ability to learn incrementally without forgetting previously acquired knowledge.

Technically, all the learning methods we have presented (except for the analytical solution for the readout weights of the ESN) are based on iterative updates of model parameters, that can be done sequentially as new data becomes available. However, these methods might suffer from the problem known as catastrophic forgetting if the statistics of the data stream they process evolve over time (i.e. are not independent and identically distributed). The continual learning setting typically involves a number of separate tasks where we assume data to be i.i.d.. The learning agent is confronted with each source of data sequentially, with no access to past sources. Using the learning methods we have described so far, based on stochastic gradient descent, a neural network automatically adapts to the new task and overwrites the model parameters that were optimized according to the previous task.

There exists a large spectrum of methods to deal with this issue. Regularization methods typically aim at reducing the forgetting by constraining learning with, for instance, sparsity constraints, early stopping, or identified synaptic weights that should not be overwritten (Jaeger, 2014a; Kirkpatrick et al., 2017). Another approach is to rely on architecture modifications when new tasks are presented, for instance by freezing some of the previously learned weights (Mallya et al., 2018), or by adding new neurons and synaptic connections to the model (Li and Hoiem, 2017). Finally, rehearsal (Rebuffi et al., 2017) and generative replay (Shin et al., 2017) methods rely on saving examples or modeling past tasks for future use. By inserting training examples from the previous tasks, either saved or replayed, into the current task, these methods allow to retrain on those data points and thus limit catastrophic forgetting. In this section, we disregard complex approaches such as rehearsal and generative replay and only consider some simple regularization or architectural techniques to improve the performance of sequence memory models in a continual learning setting.

Additionally, continual learning assumes that the learning agent is trained online while task examples are continuously brought to it by its environment. We push this online constraint further by assuming that the agent can only learn at time step t based on the currently available quantities. In our case, these quantities are the current latent variables of the models (hidden causes and states), and the observed variable $\boldsymbol{x}_t^*$. Consequently, learning methods based on BPTT do not qualify for this criterion, as they need to store in memory the past activations of the RNN hidden states to compute gradients.

To avoid confusion about the use of the word "online", we rather talk about *continual* learning to refer to the task temporality, and talk about *online* learning to refer to the target sequence (the task example) temporality. The models studied in this section are thus trained both in a continual learning setting, since the different target trajectories are provided sequentially to the agent, and using online learning mechanisms since the algorithms for learning do not rely on a memory of past neuron activations.

4.3.1 Experimental set up

Each of the model is instantiated with a hidden size $d_h = 300$, and trained sequentially on 20 sequence modeling tasks. The 20 sequences are sampled randomly from the simple data set described in section 3.4.1. We simplify the procedure by assuming that the model knows when a transition between two tasks occurs, and provide to the RNN the current task index k as a one-hot

vector input of dimension p. Otherwise, this distributional shift could for instance be automatically detected through a significant increase of the prediction error in our sequence memory modeling tasks, leading to the creation of a new neuron in the hidden causes or input layer.

The end goal of this experiment is to identify online learning mechanisms for RNNs that extend properly to the continual learning case. The RNN architectures typically comprise three types of weight parameters to be learned: the output weights, the recurrent weights and the input weights. As such, we split our analysis into three comparisons focusing on the learning methods for each type of parameters.

For each learning mechanism, we perform an hyperparameter optimization according to the method introduced in the previous section. The hyperparameters of the models are optimized in order to minimize the final average prediction error on the 20 target sequences. For each model, the hyperparameters to optimize are the learning rates associated with the input, recurrent and output weights. The score function associates each hyperparameter configuration with a real-valued score computed as the negative logarithm of the average prediction error at the end of training.

With the hyperparameter configurations we obtain, we perform for each model 10 seeds of training in the continual learning setting to measure their performances. The final average prediction error on the 20 sequences can be used to evaluate and compare the different learning mechanisms.

4.3.2 Benchmark models

The models for this benchmark were chosen in order to identify the relevant mechanisms for training RNNs in a continual learning setting. As already said, we also limited this analysis to learning algorithms that can be performed *online*, i.e. without resorting to BPTT.

For the learning of the output weights of RNNs, we identified two learning rules. First, output weights can be learned using standard stochastic gradient descent. In the RNN models we consider, the prediction $\boldsymbol{x}_t$ is not re-injected into the recurrent computations. As such, the output weights gradients can be computed using only the target signal $\boldsymbol{x}_t^*$, the prediction $\boldsymbol{x}_t$, and the hidden state $\boldsymbol{h}_t$. These computations do not involve backpropagation of a gradient through time, and thus qualify as an online learning method. This first learning rule is expressed as:

$$\boldsymbol{W}_o \leftarrow \boldsymbol{W}_o + \lambda \boldsymbol{\epsilon}_{x,t} \cdot \tanh(\boldsymbol{h}_t^{prior})^\intercal \tag{4.1}$$

The second learning mechanism that we study is stochastic gradient descent aided by Conceptors (Jaeger, 2014a). As presented in section 3.2.3, this method allows identifying a region of the hidden state space associated with each task and restricts learning on this region with minimal modifications of output weights attending to other regions. Since this method only requires to compute an estimate of the correlation matrix of reservoir states $\boldsymbol{h}_t$ during the trajectories, that can be computed iteratively, it also qualifies as an online learning method.

For the learning of the recurrent weights, we compare three learning methods inspired by PC: the PC-RNN-V, P-TNCN (Ororbia et al., 2020), and a PC-RNN-V variant that we call PC-RNN-Hebb. All three models propose the same update rule for the recurrent weights, based on the hidden state at time t and the prediction error on the hidden state layer at time $t+1$, according to the following equation:

$$\boldsymbol{W}_r \leftarrow \boldsymbol{W}_r + \lambda_r \boldsymbol{\epsilon}_{h,t+1} \cdot \tanh(\boldsymbol{h}_t^{post})^\intercal \tag{4.2}$$

The difference between the three models lies in the computation of $\boldsymbol{h}_{t+1}^{post}$. In the PC-RNN-V model, we have seen that this bottom-up computation is done using the transposed of the top-down weights used for prediction. This results in a direct minimization of VFE. In the two other models, these feedback, bottom-up weights are instead learned. In the original PC model described in (Rao and Ballard, 1997), it was proposed to learn these feedback weights using the same rule as equation 4.1 (up to a transpose to match the feedback weights shape). This learning rule ensures that with random initializations, but enough training time, the feedback weights converge to the transposed forward weights. Since this learning rule is a copy of the hebbian rule used in equation 4.1, we call PC-RNN-Hebb the RNN model using this method. The last model, P-TNCN, implements a different learning rule for the feedback weights, described by the following equation:

$$\boldsymbol{W}_b \leftarrow \boldsymbol{W}_b - \lambda_b \big(\boldsymbol{\epsilon}_{h,t} - \boldsymbol{\epsilon}_{h,t-1}\big) \cdot \boldsymbol{\epsilon}_{x,t}^\intercal \tag{4.3}$$

The model presented in (Ororbia et al., 2020) actually implements an additional term in the learning rule for the recurrent and output weights, on top of the rules explained here. This additional term led in our experiments to worse results. For this reason, we do not provide more details about this rule and turn it off during the experiments shown below.

Finally, we compare four methods to learn RNN input weights. The first two models are the PC-RNN-HC-A and PC-RNN-HC-M where we implement PC learning rules for the $\boldsymbol{W}_c$ weights, as described in section 3.3.

The third and fourth methods are respectively based on the PC-RNN-HC-A and PC-RNN-HC-M, but instead use random search to optimize the $\boldsymbol{W}_c$ weights. Our implementation of this random search is inspired by the learning algorithm proposed in (Pitti et al., 2017):

$$\boldsymbol{\delta}_i \sim \mathcal{N}(0, \sigma^2 \mathbb{I}_{d_h}) \tag{4.4}$$

$$\|\boldsymbol{\epsilon}_{x,i}\|_2 \leftarrow \text{simulate}(\boldsymbol{h}_0 + \boldsymbol{\delta}_i) \tag{4.5}$$

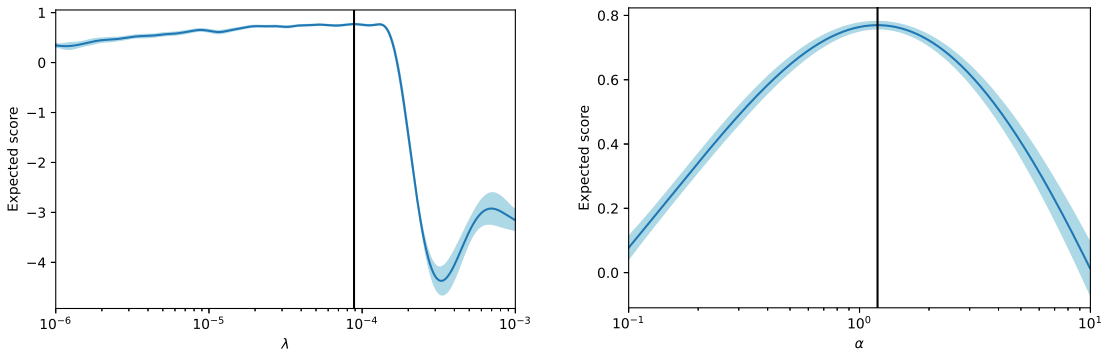
$$\boldsymbol{h}_0 \leftarrow \boldsymbol{h}_0 + \delta_i sign(\|\boldsymbol{\epsilon}_{x,i-1}\|_2 - \|\boldsymbol{\epsilon}_{x,i}\|_2) \tag{4.6}$$

where the function *sign* associates -1 to negative values, and 1 to positive values. At each training iteration i, the algorithm samples a noise vector $\boldsymbol{\delta}_i$ that is added to the initial hidden state of the RNN. After generation, the difference between the old and new average norm of the prediction error $\|\boldsymbol{\epsilon}_{x,i-1}\|_2 - \|\boldsymbol{\epsilon}_{x,i}\|_2$ is used as a measure of success of the addition of $\boldsymbol{\delta}_i$, and weights the update of the initial hidden state. Since this algorithm only relies on an average of the prediction error over the predicted sequences, that can be computed iteratively, it qualifies as an online learning algorithm.

In summary, we have identified two learning algorithm for output weights, three learning algorithms for recurrent weights, and four learning algorithms for input weights. To connect the proposed methods to the classification of continual learning methods presented above, we could categorize the Conceptors method as a regularization method, and the fact that new tasks are associated to new inputs to the RNN in the shape of hidden causes, as an architecture modification method.

4.3.3 Results

The source code for the experiments presented in this section is available on GitHub[2]. It contains our implementation of the different models as well as the hyperparameter optimization method. In appendix B, we provide the optimal hyperparameters found for each model.



(a) Score estimation of the hyperparameter optimizer with regard to the learning rate λ.

(b) Score estimation of the hyperparameter optimizer with regard to the aperture coefficient α.

Figure 4.5: Hyperparameter optimization for the Conceptors model for the continual learning comparative study. These results were obtained with $d_h = 300$.

We start by showing an example of hyperparameter optimization in figure 4.5, that was performed on the Conceptors model with $d_h = 300$. The optimized hyperparameters are the learning rate of the output weights, λ, and the aperture coefficient α. After trying 200 hyperparameter

[2]https://github.com/sino7/continual_online_learning_rnn_benchmark

configurations, the optimizer can estimate the score for all the configurations within the given range of values. These figures display the evolution of the score estimation according to λ using the optimal value for α, and according to α using the optimal value for λ. We can see that the function according to α has an inverse bell shape, while the function according to λ increases steadily before dropping once we attain values of the learning rate that no longer sustain convergence of the gradient descent.

For all the results presented below, we perform an optimization of the hyperparameters following the same protocol, and always observe score functions with similar shape and easily identifiable optimal values.

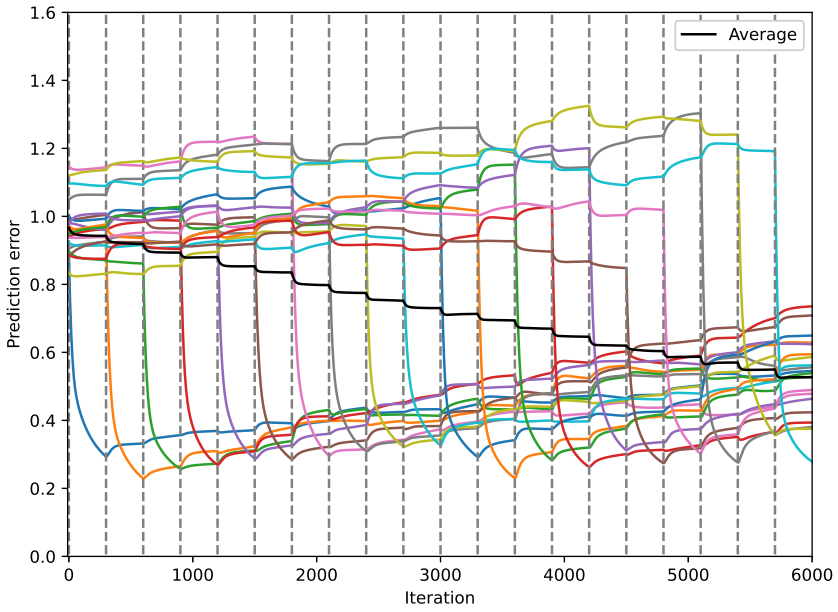


Figure 4.6: Continual learning results with the ESN model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the ESN model.

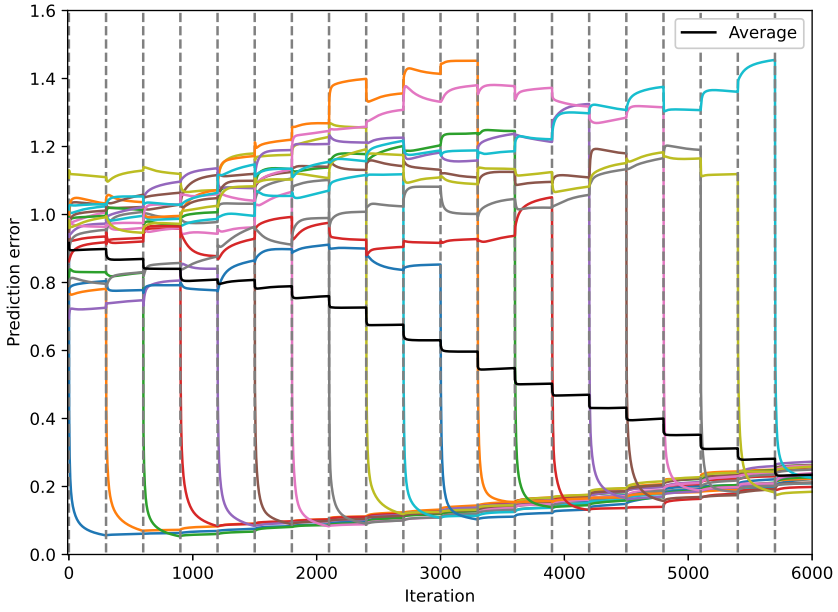


Figure 4.7: Continual learning results with the Conceptors model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the Conceptors model.

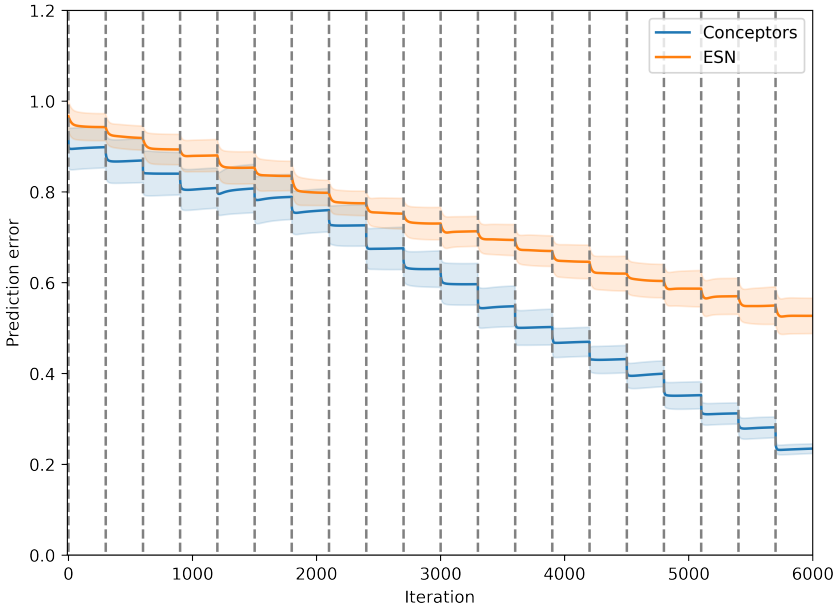


Figure 4.8: Comparison between the two learning methods for the output weights.

In figures 4.6 and 4.7, we represent the average prediction error over 10 seeds for the continual learning of 20 sequential patterns, with the hyperparameters found using the protocol described before. The vertical dashed lines in these figures delimit each of the training task. The colored lines represent the individual prediction error for each of the 20 sequence patterns (still averaged over the 10 seeds). Finally, the black line represents the average prediction error over all the sequence patterns (still averaged over the 10 seeds).

During each task, we can observe that one of the individual prediction errors decreases rapidly, while the other prediction errors only slightly change. Once the training task corresponding to a certain sequence pattern k is over, the prediction error associated to this pattern tends to increase. The better learning mechanism is the one that can limit this undesirable forgetting of previously learned sequence patterns. We can observe in figure 4.7 that the Conceptors learning mechanism limits forgetting compared to the standard stochastic gradient descent rule used in our ESN model.

At first, it can be surprising that on each individual task, the corresponding prediction error reaches a lower value for the Conceptors model than for the ESN model. In terms of learning rule, the ESN model could potentially learn each pattern with better accuracy by increasing the learning rate. However, the hyperparameter optimizer has estimated that an increased learning rate would be detrimental to the complete continual learning task. Indeed, increasing the learning rate might improve the learning on every individual task, but it would also lead to more forgetting throughout the complete task. It is only because the Conceptors learning mechanisms naturally limits forgetting that the hyperparameter optimized "allows" a higher learning rate and thus a better learning on each individual task.

We can also observe that the prediction error level that is reached during each individual task using the Conceptors model seems to increase throughout the complete task. We suppose that this is a consequence of further learning being prevented on synaptic connections associated with previous tasks' associated Conceptors. When a large number of individual tasks are over, learning is limited to synapses corresponding to a subspace of the hidden state space not belonging to any of the previous Conceptors. Increasing the aperture α would allow a better learning of the late tasks, but at the detriment of an increased forgetting of the early tasks.

Figure 4.8 compiles these previous figures to compare the average prediction error using both learning mechanism. At the end of training, we can see that the Conceptors model achieve a significantly lower prediction error than the ESN using the standard stochastic gradient descent rule for the learning of the output weights. Based on this first experiment, we can conclude that the Conceptors learning method is more suited to train the output weights of an RNN than standard stochastic gradient descent.

In this second experiment, we compare the PC-RNN-V with two variants using learning rules

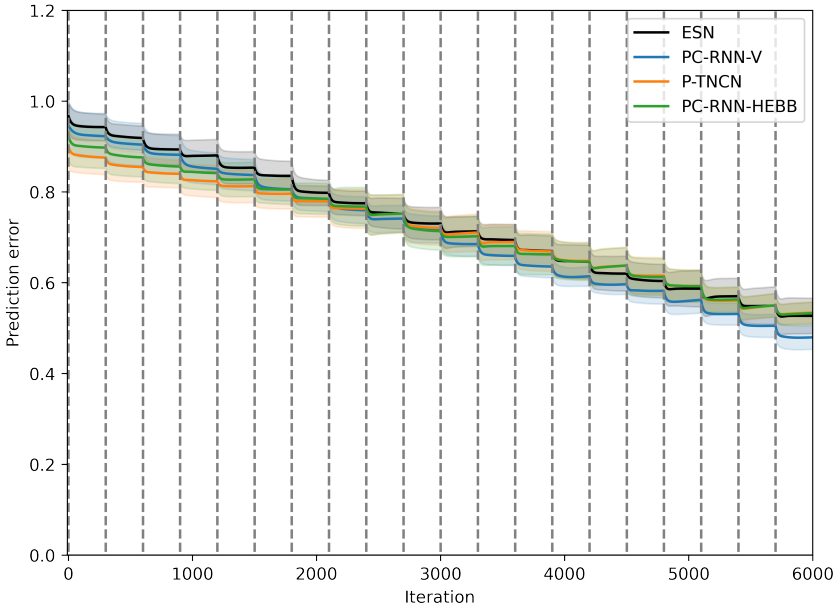


Figure 4.9: Comparison between the three learning methods for the recurrent weights, and the ESN model where no learning is performed on the recurrent weights.

for the feedback weights instead of using the transposed feedforward weights. These three learning methods in the end provide different update rules for the recurrent weights of the RNN. The results of this second comparative analysis are provided in figure 4.9. We have added to the comparison the evolution of the average prediction error with no recurrent weights learning (in black). We can observe that the P-TNCN and the PC-RNN-Hebb models perform exactly the same as the model with no recurrent weights learning. In fact, this can easily be explained when looking at the hyperparameters that were found by the optimizer for these models. In both cases, the hyperparameter optimization suggested using the lowest possible (within the available range) value for the learning rate of the recurrent weights. Empirically, the optimizer has found that in these models it was better to cancel the recurrent weights learning, since it led to a catastrophic forgetting that was not compensated by the improvement on each individual task. Consequently, with very low values for the recurrent weights learning rate, the models are roughly equivalent to the ESN model.

The PC-RNN-V however shows a slight improvement in average prediction error compared to the model with no recurrent learning. Learning recurrent weights using the PC-RNN-V learning rule is still beneficial, but does not significantly improve the model's performances. Later, we will try to combine this learning rule for the recurrent weights with the Conceptors learning rule for the output weights. We can still conclude based on these results that recurrent weights learning in a continual learning setting is difficult and might often lead to more catastrophic forgetting.

Figure 4.10 displays the results obtained with the four learning mechanisms for input weights, and the PC-RNN-V as a baseline. If we consider the case where the learning of the weights $\boldsymbol{W}_c$ is turned off in the PC-RNN-HC-A/M models, we obtain a model that is equivalent to the PC-RNN-V. For this reason, we can use the PC-RNN-V model as a baseline to measure the improvement brought by the learning in the input layer.

These results suggest that the learning methods using random search (RS suffix) perform poorly corresponding to the learning rules relying on propagation of error using PC. The two models using random search perform similarly to the baseline PC-RNN-V model. This observation is surprising, since the $\boldsymbol{W}_c$ weights in PC-RNN-HC-A/M architectures are directly factored according to each individual task. Indeed, during the task k, we can limit learning on the k-th column of the $\boldsymbol{W}_c$ weights, since these are the only weights that influence the RNN trajectory. Consequently, training this layer should not cause any additional forgetting, and thus should only bring improvements over the baseline PC-RNN-V model. Since the two models using random search did not bring any improvement, we suppose that this is due to the limited number of iterations allowed for the training on each individual task. We observed that in general training with random search as in the

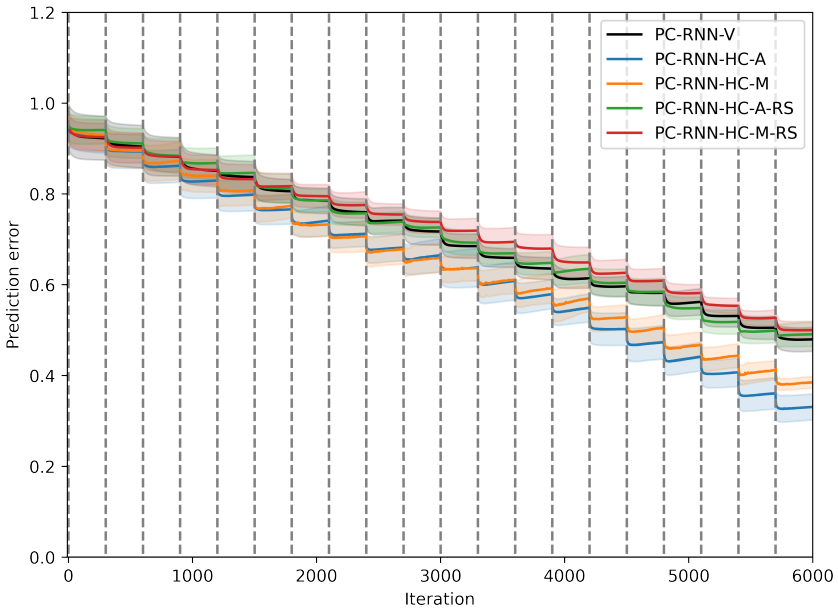


Figure 4.10: Comparison between the three learning methods for the input weights, and the PC-RNN-V model with only output and recurrent weights learning.

INFERNO model (Pitti et al., 2017) needed many more iterations than gradient-based methods.

The PC-RNN-HC-A/M models trained using the PC-based learning rules still showed some significant improvement compared with the PC-RNN-V baseline, with the PC-RNN-HC-A model performing slightly better than the PC-RNN-HC-M model. This experiment allow us to conclude that learning rule for input weights proposed by the PC-RNN-HC-A model is the most suited to a continual learning setting.

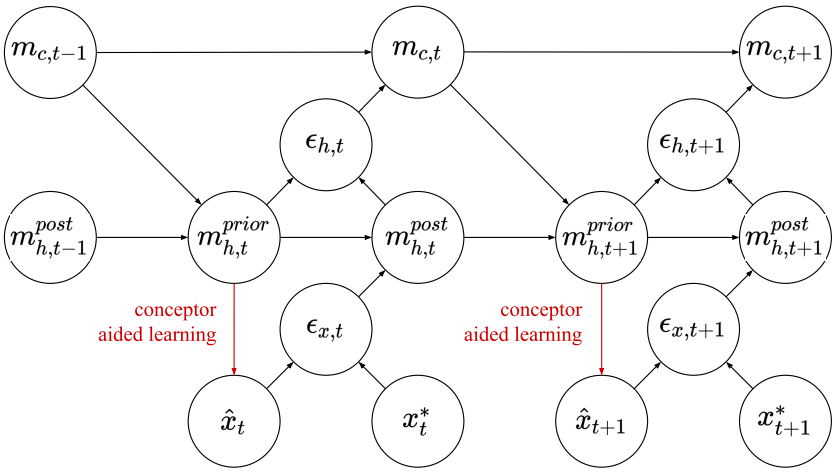


Figure 4.11: PC-RNN-HC-A model with Conceptors aided learning.

Finally, we can inquire whether these different learning mechanisms combine well with each other. We implement the Conceptors learning rule on the output weights of a PC-RNN-HC-A model, as represented in figure 4.11. Figure 4.12 displays the prediction error on each individual task as well as the average prediction error throughout learning, using this model. Interestingly, virtually no forgetting seems to happen during learning, as the individual prediction errors plateau after decreasing during the corresponding individual tasks.

Additionally, the hyperparameter optimizer in this case recommended using the lowest possible value for the recurrent weights learning rate. This suggests that the recurrent weights learning negatively interferes with the Conceptors model. The Conceptors model might be sensible to

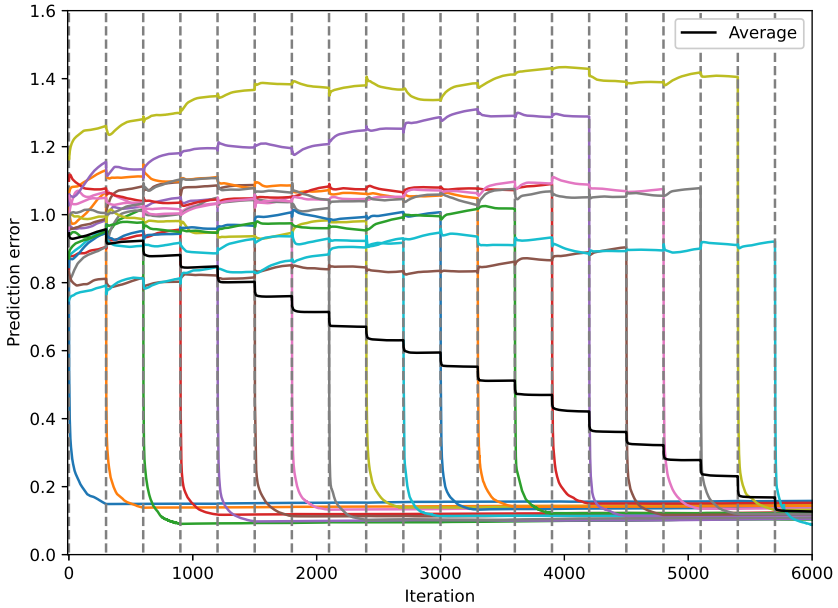


Figure 4.12: Continual learning results using the PC-RNN-HC-A model with Conceptors. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-HC-A model with Conceptors.

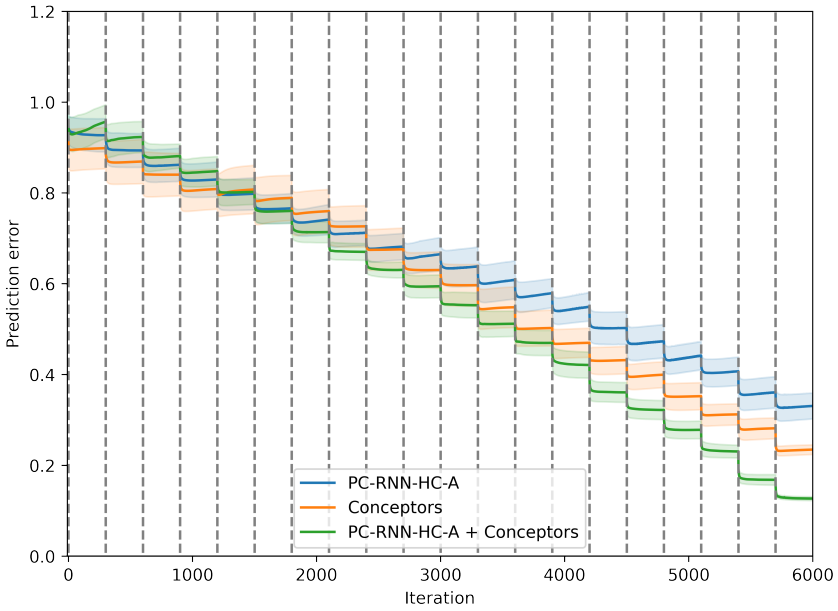


Figure 4.13: Comparison between the Conceptors model, the PC-RNN-HC-A model, and the variation of the PC-RNN-HC-A model using Conceptors to learn the output weights.

recurrent weights learning, since this could turn the previously learned Conceptors into obsolete descriptors of the corresponding hidden state trajectories.

We compare these results with both the Conceptors and the PC-RNN-HC-A model in figure 4.13, which confirms that this combination of online learning methods seems to provide the sequence memory model the best suited for continual learning. Overall, this study suggests that regularization methods such as Conceptors, and architectural methods as proposed in the PC-RNN-HC architectures, can help designing RNN models with online learning rules suitable for continual learning.

4.4 Conclusion

In this chapter, we have conducted two comparative studies to evaluate the sequence memory models, according to their generative capacity, and their possible use in a continual learning setting.

We have seen in chapter 2 that there were some connections between the BP learning algorithm and the PC-based learning algorithm. Both BP and PC also provide inference mechanisms allowing to update the agent's internal belief based on the prediction error. Using the PC learning rules has several advantages. First, it does not require storing past activations in order to compute gradients, which decreases its computational cost and allow this method to be performed online. Second, as a variational Bayes method, it takes into account a prior distribution over latent variables. This is one of the reasons why VAEs, even though they are trained with BP, have had so much success at representation learning. The effect of the prior distribution is considered during learning as part of the loss function, and consequently, the recognition density in VAEs is constrained to *look like* the prior distribution. This plays an important role in obtaining disentangled representations. Still, these advantages might not compensate for the large performance gap observed using BP-based and PC-based learning rules.

For PC-based learning to contend with BP, we would need to either improve the convergence speed of the PC-based learning or decrease the computational cost of this method. A direction to improve the learning algorithm might be to study the effect of the precision coefficients. These coefficients directly scale the error signal being propagated bottom-up, and could play a role in improving the convergence speed of PC-based learning. To decrease the computational cost of PC, it could be interesting to implement these algorithms on dedicated hardware reproducing the connection patterns involved in the PC theory.

Chapter 5

Memory retrieval

5.1 Introduction

In this chapter, we address the question of memory retrieval within the sequence memory models derived previously. Sequence memories implement a function $f : \boldsymbol{c} \rightarrow (\boldsymbol{x}_1, \cdots \boldsymbol{x}_T)$. Based on p (key, sequence) pairs, these models can be trained in a supervised manner to associate each key $\boldsymbol{c}_k$ to the corresponding sequence $(\boldsymbol{x}_1^*, \cdots \boldsymbol{x}_T^*)_k$. In the models derived in the last chapter, training over the p trajectories ensures the correspondence between the k sequential patterns and the k hidden causes values. We have used a one-hot embedding for the hidden causes such that the k-th hidden causes value is the one-hot vector of dimension p activated on the k-th dimension. In this chapter, we also study the possibility of relaxing this assumption and authorizing the hidden causes values to be inferred during training.

During memory retrieval, the task is to retrieve the sequence pattern written in memory that best fits a provided target pattern, that we can also call *query*. Based on the models we have previously derived, we suggest splitting this retrieval process into an inference step and a generation step. The inference step attempts to retrieve the key associated with the target sequence, and the generation step regenerates the learned sequence based on this retrieved key. The success of this memory retrieval process depends on the quality of the mechanism allowing to retrieve keys from target patterns.

This chapter is organized as follows. In section 5.2, we review different ways to approach this problem that have been proposed in the literature. In section 5.3, we derive memory retrieval mechanisms based on the PC-RNN-HC models presented in the last chapter. In section 5.4, we present some experiments performed using these mechanisms, and finally we discuss our results in section 5.5.

5.2 Related work

5.2.1 Chaotic itinerancy

One field of research that can be related to the question of memory retrieval is the phenomenon known as Chaotic Itinerancy (CI) in the dynamical systems literature.

CI describes the behavior of large non-linear dynamical systems consisting of chaotic transitions between quasi-attractors (Tsuda, 1991; Kaneko and Tsuda, 2003). It was first observed in a model of optical turbulence (Ikeda et al., 1989), using globally coupled maps in a chaotic system (Kaneko, 1990) and in high dimensional neural networks (Tsuda, 1991). From a neuroscientific point of view, this phenomenon is interesting as such systems exhibit complex behaviors that usually require a hierarchical structure in neural networks. Studying CI could help better understanding the mechanisms responsible for the emergence of structure in large populations of neurons.

In cognitive neuroscience, it is believed that attractors or quasi-attractors could represent perceptual concepts or memories, and that cognitive processes such as memory retrieval or thinking would require neural trajectories transitioning between such attractors. CI is also gaining interest in neurorobotics, as it allows designing agents with the ability to autonomously switch between different behavioral patterns without any external commands. Several studies have tried to model

CI with learned attractor patterns. (Yamashita and Tani, 2008; Namikawa et al., 2011) propose a method where this functional structure emerges from an MTRNN. Sequential patterns are encoded in a rapidly varying recurrent population while another population with a longer time constant controls transitions between these patterns. (Inoue et al., 2020) models CI, using RC techniques, with the interplay between an input RNN and a chaotic RNN where desired patterns have been learned with innate trajectory training (Laje and Buonomano, 2013).

A naive memory retrieval technique could thus consist in implementing CI in our RNN models. The hidden causes could randomly alternate between the different keys that have been learned. This itinerancy would randomly try out different hidden causes values (i.e. keys) until finding the one that correctly generates the provided target sequence (i.e. the query).

5.2.2 Memory retrieval

Several approaches have been proposed in the literature to design memory models equipped with retrieval mechanisms.

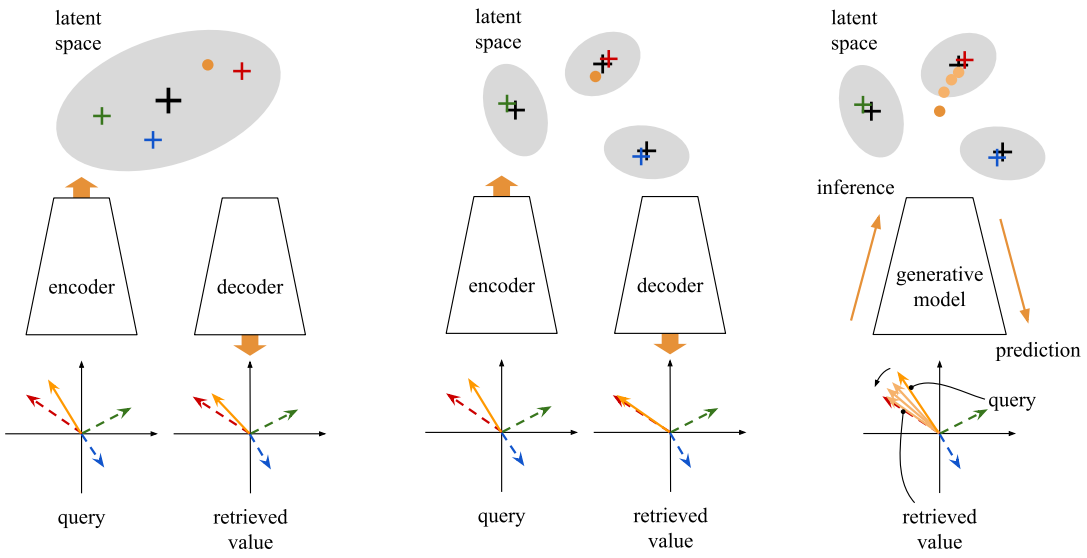
The Hopfield network is an associative memory model that was introduced in (Hopfield, 1982). This model is composed of binary neurons (i.e. neurons that can either be idle or activated) with recurrent connections. At each time step, the activations of the neurons are calculated based on these recurrent connections and a threshold value. The connection weights can be trained so that the network dynamics always converge to desired patterns. The memory retrieval consists in initializing the network with a query pattern, and iteratively running the update rule until the network converges to one of the trained patterns.

These models have been extended to work on continuous signals as well (Demircigil et al., 2017). The network can be trained so that certain continuous multidimensional patterns correspond to attractors of the dynamical system. A recent work (Ramsauer et al., 2020) draws a connection between these modern Hopfield networks and the attention mechanisms of Transformers (Vaswani et al., 2017). They propose an update rule for continuous Hopfield networks and show how this rule relates to the attention mechanism. The recurrent network can be seen as computing attention weights with regard to a collection of keys at each time step. The network activation can reach several equilibrium configurations when initialized with a query pattern. One of the types of possible equilibrium configurations is when the network activation has reached one of the patterns written in the network (as in standard Hopfield networks). However, the dynamics in these networks can also converge to metastable states corresponding to averages of subsets of the learned patterns.

Other approaches such as Neural Turing Machines (NTMs) (Graves et al., 2014) and Differentiable Neural Computers (DNCs) (Graves et al., 2016) also use attention mechanisms to read inside a memory store. However, contrary to the Hopfield network, the reading process is instantaneous and does not need the convergence of a dynamical system. Additionally, we can note that in Hopfield networks there is no distinction between the key (i.e. the address of the stored pattern) and the value (i.e. the stored pattern). The key *is* the value, meaning that addressing in this memory can only be done based on its content. As such, it can be classified as a content-based addressing memory. In contrast, in NTMs, patterns (values) are stored with associated keys. A query is compared to the different keys to retrieve the correct patterns in memory. For comparison, the approach proposed in this chapter combines both ideas, since the retrieval of the correct key is an iterative process as in Hopfield networks, and the key is then used to regenerate the sequence pattern (the value), as in NTMs.

These models do not really fit into the PC framework. Closer to our approach are the models based on variational Bayes methods, such as VAEs (Kingma and Welling, 2014; Rezende et al., 2014), already presented in the previous chapter. Using VAEs, the problem of memory retrieval can be seen as that of encoding the provided pattern (the query) into a latent representation (the key), and using this latent representation to regenerate the correct pattern (the value). However, it is not possible to embed preferences for certain keys in the encoding process. The representation is continuous and there is no mechanism pulling its value towards certain keys written in the memory. Figure 5.1a illustrates the use of a VAE for memory retrieval. A query pattern is provided to the encoder, that infers a corresponding key in the latent space. This key is given to the decoder that generates the retrieved pattern.

Some models augment regular VAEs with external memory stores conditioning the latent variable in the generative models. In (Wu et al., 2018; Wu et al., 2018) the authors suggest computing the latent variable (the key) as a linear combination of vectors recorded in a memory matrix. The



(a) VAE (b) GMVAE (c) Our method

Figure 5.1: Variational Bayes models for memory retrieval. In all figures, the gray ellipses represent the prior distribution on the latent representation. The red, green and blue crosses represent the keys corresponding to the pattern vectors of the same color represented on the bottom of the figures. They query and the retrieved values are represented by orange vectors. The inferred key is represented as an orange dot in the latent space.

weights of this linear combination are adapted iteratively until convergence. (Bornschein et al., 2017) propose a similar scheme where the weights of the linear interpolation are computed based on a similarity measure between the provided query and the available keys.

All these VAE models consider a continuous representation space, while we are interested in inferring a discrete value, more precisely, the index of the key in the sequence memory. In regular VAEs, the prior distribution over the latent variable is a Gaussian. This assumption causes the representation manifold to be composed of one cluster. (Dilokthanakul et al., 2016) propose an extension of VAEs, names Gaussian Mixture VAE (GMVAE), using a Gaussian mixture prior distribution onto the latent variable. The model can be trained by combining the reparameterization trick (used in regular VAEs) with Monte Carlo sampling (to account for the possible values of the discrete random variable). This method allows having a representation manifold composed of many clusters. The trained encoder can then be used to infer the category (the key) of a given target pattern (the query). The method proposed in this chapter also attempts at performing variational inference using a Gaussian mixture prior on the keys, but using the PC framework. Figure 5.1b illustrates the use of the GMVAE for memory retrieval. The Gaussian mixture prior should enable the autoencoder to generate patterns closer to the ones written in memory (represented in red, green, and blue).

Finally, all the methods we have reviewed here do not consider the possibility of storing and retrieving sequential patterns. Still, the VAE models could extend to this case by using two RNN instances for the encoder and the decoder. The encoder RNN would compute the key associated with a given sequential query, and the decoder RNN would generate the sequence pattern corresponding to the retrieved key.

Since this thesis focuses on PC-based models, we attempt in the next section to derive a memory retrieval mechanism relying on the bottom-up inference mechanism of PC. We combine this inference mechanism with a Gaussian mixture prior distribution onto the hidden causes (which fill the role of the keys in the PC-RNN-HC models). Our proposed model can be illustrated by the figure 5.1c. Based on the initial query, the iterative message-passing scheme induced by PC combined with the Gaussian mixture prior enables retrieval of the correct pattern from the memory. For simplicity, we have represented 2D patterns in these figures. For a sequence memory, these patterns would be sequences of possibly high-dimensional vectors.

5.3 Methods

In this section, we derive a memory retrieval algorithm inspired by the free-energy formulation of PC. This algorithm exploits a prior probability distribution on the hidden causes in order to obtain hidden causes dynamics that converge to one of the learned keys in the sequence memory. We present two variations of this algorithm, *unstructured* and *structured*, that differ in whether they authorize the keys of the sequence memory to be optimized during learning.

5.3.1 Prior distribution on the hidden causes

All the PC-RNN variants with hidden causes that we derived in chapter 3 do not take into account any prior distribution upon the hidden causes $\boldsymbol{C}$. However, as a model implementing a form of variational Bayes inference upon latent variables $\boldsymbol{H}$ and $\boldsymbol{C}$, the model's approximate posterior belief upon latent variables should depend on its prior belief. In the derivations of these models, we proposed to use the recurrent computations as a mechanism to compute the prior distribution over $\boldsymbol{H}$ at time t. The iterative algorithm resulting from this choice of prior distribution aligns nicely with standard implementations of RNNs. However, we have disregarded the prior distribution on $\boldsymbol{C}$ using the assumption that this prior distribution was a very flat Gaussian with no impact on the inference computations. In this section, we go back on this assumption and instead suggest using this prior distribution over hidden causes in order to guide the retrieval of learned sequential patterns. The probabilistic model of the PC-RNN-HC models is represented in figure 5.2.

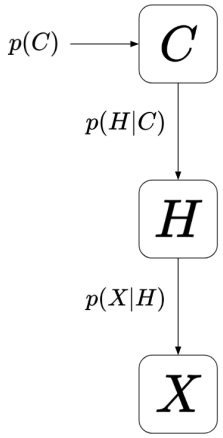


Figure 5.2: Probabilistic model of the PC-RNN-HC models.

We start from the expression of the VFE corresponding to this generative model. Since for now we have not made any assumption about the form of the prior distribution $p(\boldsymbol{C})$, we simply express the VFE as:

$$\begin{aligned} F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c) = &- \log p(\boldsymbol{x}^*|\boldsymbol{m}_h) \\ &- \log p(\boldsymbol{m}_h|\boldsymbol{m}_c) \\ &- \log p(\boldsymbol{m}_c) \\ &+ C' \end{aligned} \tag{5.1}$$

where C' is a constant (not to be confused with the random variable $\boldsymbol{C}$) and the conditional distribution $p(\boldsymbol{h}|\boldsymbol{c})$ is made dependent on the past hidden state $\boldsymbol{h}_{past}$ through recurrent connections. As usual, the conditional distributions $p(\boldsymbol{h}|\boldsymbol{c})$ and $p(\boldsymbol{x}|\boldsymbol{h})$ are assumed to take a Gaussian form:

$$p(\boldsymbol{h}|\boldsymbol{c}) = \mathcal{N}(\boldsymbol{h}; \boldsymbol{f}(\boldsymbol{c}, \boldsymbol{h}_{past}), \sigma_h^2 \mathbb{I}_{d_h}) \tag{5.2}$$

$$p(\boldsymbol{x}|\boldsymbol{h}) = \mathcal{N}(\boldsymbol{x}; \boldsymbol{g}(\boldsymbol{h}), \sigma_x^2 \mathbb{I}_{d_x}) \tag{5.3}$$

We have already seen that using the Gaussian assumption for $p(\boldsymbol{h}|\boldsymbol{c})$ and $p(\boldsymbol{x}|\boldsymbol{h})$, the VFE could be expressed as:

$$F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c) = \frac{1}{2\sigma_x^2}\|\boldsymbol{x}^* - \boldsymbol{g}(\boldsymbol{m}_h)\|_2^2 + \frac{d_x}{2}\log(\sigma_x^2) \tag{5.4}$$

$$+ \frac{1}{2\sigma_h^2}\|\boldsymbol{m}_h - \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})\|_2^2 + \frac{d_h}{2}\log(\sigma_h^2) \tag{5.5}$$

$$- \log p(\boldsymbol{m}_c) \tag{5.6}$$

$$+ C' \tag{5.7}$$

Finally, contrary to the PC-RNN-HC models derived previously where we ignored the prior distribution on $\boldsymbol{C}$, here we propose to use a mixture of Gaussians distribution. We can use a mixture of Gaussians to embed the preference of certain values for $\boldsymbol{C}$ in the model. Indeed, we will see that this prior distribution pulls the inference process towards the means of the Gaussian mixture.

$$p(\boldsymbol{m}_c) = \sum_{k=1}^{n} \pi_k \mathcal{N}(\boldsymbol{m}_c; \boldsymbol{\mu}_k, \sigma_c^2 \mathbb{I}_{d_c}) \tag{5.8}$$

This mixture model is composed of n multivariate Gaussians of means $\boldsymbol{\mu}_k$ and covariance matrices $\sigma_c^2\mathbb{I}_{d_c}$, weighted by mixing coefficients π_k. This distribution can be seen as the composition of a discrete probability distribution $p(Z = z_k) = \pi_k$ on a latent discrete variable Z, and the Gaussian probability distributions $p(\boldsymbol{m}_c|Z = z_k) = \mathcal{N}(\boldsymbol{m}_c; \boldsymbol{\mu}_k, \sigma_c^2\mathbb{I}_{d_c})$.

Interestingly, the direct computation of $\log p(\boldsymbol{m}_c)$ is tractable, since it only requires summing over n Gaussian probabilities. In chapter 6, we study another case of mixture probabilistic model where this time we also perform variational inference on the discrete latent variable Z. The variational inference computations depend on the constraint we impose on the recognition density $q(Z)$. In chapter 6, we choose an extreme recognition density such that $q(Z)$ is equal to 1 for one of the possible values of Z, and 0 otherwise. For the question of memory retrieval, this assumption is too constraining and would hinder the retrieval process. Another option for recognition density would be to consider a categorical distribution parameterized by coefficients $q_k = q(Z = z_k)$. Since this distribution is as expressive as the true posterior distribution, the resulting computations would be equivalent in terms of complexity, and only approximations of the more accurate algorithm we present here.

The iterative algorithm for inference can be obtained by computing the derivatives of the VFE with regard to the recognition density means $\boldsymbol{m}_h$ and $\boldsymbol{m}_c$. Since nothing has changed for the recognition density means $\boldsymbol{m}_h$, we only focus here on $\boldsymbol{m}_c$:

$$\begin{aligned}\frac{\partial F(\boldsymbol{x}^*, \boldsymbol{m}_h, \boldsymbol{m}_c)}{\partial \boldsymbol{m}_c} = &- \frac{1}{\sigma_h^2}\big(\boldsymbol{m}_h - \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big) \cdot \frac{\partial \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})}{\partial \boldsymbol{m}_c} \\ &- \frac{d \log p(\boldsymbol{m}_c)}{d\boldsymbol{m}_c}\end{aligned} \tag{5.9}$$

The first term of this gradient corresponds to the bottom-up signal coming from the layer below, as already shown in the previous chapter. The derivations of this term depend on the definition of the function $\boldsymbol{f}$, that can model either an additive or multiplicative influence of the hidden causes onto the hidden state dynamics. The second term of this gradient corresponds to the influence of

the prior distribution on $\boldsymbol{C}$. We can develop this derivative as:

$$\begin{aligned}
\frac{d \log p(\boldsymbol{m}_c)}{d\boldsymbol{m}_c} &= \frac{\frac{dp(\boldsymbol{m}_c)}{d\boldsymbol{m}_c}}{p(\boldsymbol{m}_c)} \\
&= \frac{1}{p(\boldsymbol{m}_c)} \sum_{k=1}^{n} \pi_k \frac{1}{(2\pi\sigma_c^2)^{\frac{d_c}{2}}} \frac{d \exp\big\{ - \frac{\|\boldsymbol{m}_c - \boldsymbol{\mu}_k\|_2^2}{2\sigma_c^2} \big\}}{d\boldsymbol{m}_c} \\
&= -\frac{1}{p(\boldsymbol{m}_c)} \sum_{k=1}^{n} \pi_k \frac{1}{(2\pi\sigma_c^2)^{\frac{d_c}{2}}} \frac{\boldsymbol{m}_c - \boldsymbol{\mu}_k}{\sigma_c^2} \exp\big\{ - \frac{\|\boldsymbol{m}_c - \boldsymbol{\mu}_k\|_2^2}{2\sigma_c^2} \big\} \\
&= -\sum_{k=1}^{n} \pi_k \frac{\mathcal{N}(\boldsymbol{m}_c; \boldsymbol{\mu}_k, \sigma_k^2)}{p(\boldsymbol{m}_c)} \frac{(\boldsymbol{m}_c - \boldsymbol{\mu}_k)}{\sigma_c^2} \\
&= -\sum_{k=1}^{n} p(Z = z_k | \boldsymbol{m}_c) \frac{\boldsymbol{m}_c - \boldsymbol{\mu}_k}{\sigma_c^2}
\end{aligned} \tag{5.10}$$

where $p(Z = z_k|\boldsymbol{m}_c)$ is the true posterior probability distribution on Z knowing $\boldsymbol{m}_c$. We can indeed verify that:

$$\frac{\pi_k \mathcal{N}(\boldsymbol{m}_c; \boldsymbol{\mu}_k, \sigma_k^2)}{p(\boldsymbol{m}_c)} = \frac{p(Z = z_k) p(\boldsymbol{m}_c | Z = z_k)}{p(\boldsymbol{m}_c)} = p(Z = z_k | \boldsymbol{m}_c) \tag{5.11}$$

We can now derive the iterative update rule on $\boldsymbol{m}_c$ as:

$$\begin{aligned}
\boldsymbol{m}_c \leftarrow \boldsymbol{m}_c + & \underbrace{\frac{\alpha}{\sigma_h^2} \big(\boldsymbol{m}_h - f(\boldsymbol{m}_c, \boldsymbol{h}_{past})\big) \cdot \frac{\partial \boldsymbol{f}(\boldsymbol{m}_c, \boldsymbol{h}_{past})}{\partial \boldsymbol{m}_c}}_{\text{Bottom-up}} \\
& \underbrace{- \frac{\alpha}{\sigma_c^2} \big(\boldsymbol{m}_c - \sum_{k=1}^{n} p(z_k|\boldsymbol{m}_c) \boldsymbol{\mu}_k\big)}_{\text{Top-down}}
\end{aligned} \tag{5.12}$$

where α is the update rate. The first term of this update rule pulls $\boldsymbol{m}_c$ towards values that minimize the prediction error on the hidden state layer. The second term of this update pulls $\boldsymbol{m}_c$ towards the weighted average of the Gaussian mixture means $\boldsymbol{\mu}_k$, the weights corresponding to the true posterior distribution on Z knowing $\boldsymbol{m}_c$. This update rule can be included in the PC-RNN-HC-A and PC-RNN-HC-M models. This mechanism should guide the inference of $\boldsymbol{m}_c$ towards the values that are probable under the prior distribution.

Compared to the PC-RNN-HC models, we have only added a term influencing the temporal evolution of $\boldsymbol{m}_c$.

5.3.2 Mechanisms influencing the hidden causes dynamics.

The hidden causes dynamics are influenced by the bottom-up prediction error signal, and the top-down prior distribution. On top of these two mechanisms, we propose to add some noise in the hidden causes update rule. At each time step, we sample an additive noise from a multivariate Gaussian distribution centered on 0 and of variance σ_r^2. If we were to only consider this mechanism, the hidden causes would exhibit a random walk trajectory parameterized by σ_r. We speculate that applying this noise onto the hidden causes dynamics might help avoiding local minima of VFE in which the gradient descent process might get stuck. In conclusion, we consider three mechanisms influencing the dynamics of the hidden causes:

- **Inference:** The bottom-up influence of the prediction error signal, parameterized by the coefficients $\alpha_h = \frac{\alpha}{\sigma_h^2}$ and $\alpha_x = \frac{\alpha}{\sigma_x^2}$.
- **Prior distribution:** The top-down influence of the prior distribution, parameterized by the coefficients α and σ_c.
- **Random walk:** The additive noise, parameterized by the standard deviation σ_r.

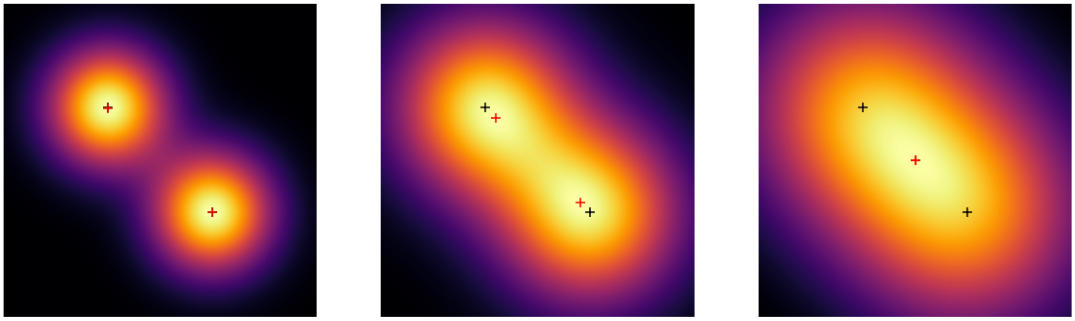
5.3.3 Unstructured case

In this section, we describe the effect of the mechanisms introduced above onto the dynamics of the hidden causes.

We first assume that the hidden causes $\boldsymbol{m}_c$ lie in a $d_h = n$ dimensional space where n is the number of modes of the Gaussian mixture model, and also equal to the number of sequence patterns p the model has learned. The means of the Gaussian mixture model $\boldsymbol{\mu}_k$ correspond to the previously learned hidden causes vectors, i.e. the keys of the sequence memory. For each sequence pattern k, the associated hidden causes representation is assumed to be the one-hot vector activated on the k-th dimension. During training on each sequence pattern k, this one-hot vector value was forced onto $\boldsymbol{m}_c$ during the complete trajectory, and the inference of the hidden states and causes was turned off by setting α_x and α_h to 0. At the end of training, using the values $\boldsymbol{\mu}_k$ as initial hidden causes in generation mode yields trajectories very close the data set target patterns.

Because of the choice of using one-hot representations, the p vectors $\boldsymbol{\mu}_k$ are all orthogonal and constitute an orthonormal basis of the hidden causes space. If $d_c = 3$, the three means $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$ and $\boldsymbol{\mu}_3$ form an equilateral triangle in a 3D space. If $d_c = 4$, the four means $\boldsymbol{\mu}_1$, $\boldsymbol{\mu}_2$, $\boldsymbol{\mu}_3$ and $\boldsymbol{\mu}_4$ form an equilateral tetrahedron in the 4D space. We can see this configuration of the mixture density means as *unstructured* since the relative similarity of the data set sequence patterns corresponding to each $\boldsymbol{\mu}_k$ is not reflected in the values of $\boldsymbol{\mu}_k$.

In opposition, a structured configurations would use values for $\boldsymbol{\mu}_k$ that reflect the relative similarities between the corresponding target patterns $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)_k$. For instance, if we learn trajectory patterns corresponding to the letters a, b and d, we would expect the density means corresponding to a and d to be closer because of the similarity between the two sequential patterns.



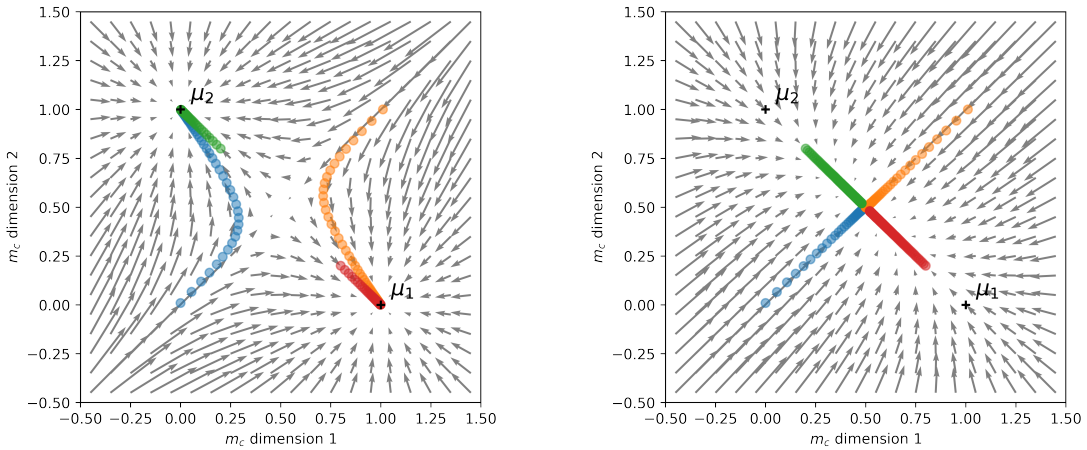
(a) $\sigma_c = 0.4$ (b) $\sigma_c = 0.6$ (c) $\sigma_c = 0.8$

Figure 5.3: Gaussian mixture probability distributions with $n = d_h = 2$. The Gaussians centers $\boldsymbol{\mu}_0 = (1, 0)$ and $\boldsymbol{\mu}_1 = (0, 1)$ are represented in black. The red points represent the minima of the distributions. The prior means $\boldsymbol{\mu}_k$ correspond to the one-hot vectors activated on the k-th dimension, and the mixture coefficients π_k are set uniformly : $\pi_k = 1/n$.

As represented in figure 5.3, the parameter σ_c determines the shape of the prior distribution on hidden causes. With low values of σ_c, the second term in equation (5.12) pulls the hidden causes variable towards one of the prior means $\boldsymbol{\mu}_k$. These values for $\boldsymbol{m}_c$ correspond to temporal dynamics that have previously been trained to match each of the desired sequence pattern. With high values of σ_c, the Gaussians merge into a concave function with a global maximum corresponding to the average of all the prior means $\boldsymbol{\mu}_k$. In this situation, the second term in equation (5.12) pulls the hidden causes recognition density mean $\boldsymbol{m}_c$ towards this average value, for which no training was performed.

In this chapter, we assume that the memory model has already been trained. In our experiments, the neural network is initialized with hidden causes values $\boldsymbol{m}_{c,0}$ that can be random, and is provided a target trajectory $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$. The prior distribution $p(\boldsymbol{m}_c)$, parameterized by σ_c, the bottom-up inference signal originating from the prediction error between the prediction $\boldsymbol{x}_t$ and target $\boldsymbol{x}_t^*$, as well as an additive noise, all have an influence on the dynamics of $\boldsymbol{m}_c$.

Figure 5.4 represents as vector fields the influence of the Gaussian mixture prior probability, in a simplified set up where $n = p = d_c = 2$. Since the hidden causes space is of dimension 2, we can display the dynamics of $\boldsymbol{m}_c$ induced by the different mechanisms we have described, for four



(a) Dynamics of $\boldsymbol{m}_c$ dictated by the prior probability with $\sigma_c = 0.4$.

(b) Dynamics of $\boldsymbol{m}_c$ dictated by the prior probability with $\sigma_c = 0.8$.

Figure 5.4: Influence of the prior probability distribution on the dynamics of $\boldsymbol{m}_c$.

different initializations $\boldsymbol{m}_{c,0}$ (represented in blue, orange, green and red).

These simulations were obtained using a PC-RNN-HC-A model trained with two sequence patterns corresponding to the letters a and b from the handwriting data set. The prior probability was computed with $\sigma_c = 0.4$ in figure 5.4a and $\sigma_c = 0.8$ in figure 5.4b. These figures display a vector field showing the direction and amplitude of the gradient in all the positions of the hidden causes space (here of dimension 2). Additionally, we represented four possible hidden causes trajectories starting from different initial points $\boldsymbol{m}_{c,0}$. The blue and orange initial hidden causes are close to the boundary between the two attractors induced by the prior probability with $\sigma_c = 0.4$. On the opposite, the red and green initial hidden causes are chosen close respectively to the first mixture density mean $\boldsymbol{\mu}_1$ and second mixture density mean $\boldsymbol{\mu}_2$.

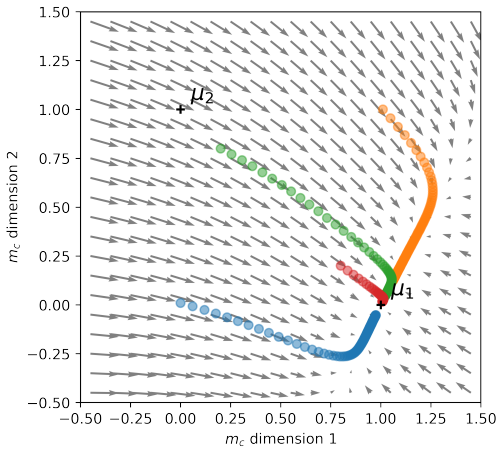
If we only consider the influence of the prior distribution, the dynamics of $\boldsymbol{m}_c$ are directly attracted either by the nearest mixture density mean when $\sigma_c = 0.4$ (figure 5.4a), or by the average of the mixture density mean when $\sigma_c = 0.8$ (figure 5.4b). We consider two possible tasks using this *unstructured* configuration:

Itinerancy: We suppose that by alternating between phases where σ_c is low and phases where σ_c is high, the dynamics of $\boldsymbol{m}_c$ would periodically return to the global attractor when σ_c is high, and randomly be directed towards the attractors $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_2$ when σ_c is low. Interestingly, the learned hidden causes configurations $\boldsymbol{m}_c = \boldsymbol{\mu}_1$ and $\boldsymbol{m}_c = \boldsymbol{\mu}_2$ can correspond to certain cycle attractors for the dynamics of $\boldsymbol{m}_h$. In this case, $\boldsymbol{m}_h$ could exhibit random attractor switching dynamics similar to CI. In this set up, we do not assume that a target trajectory $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$ is provided, and the bottom-up prediction error signal influencing the dynamics of $\boldsymbol{m}_c$ is removed.

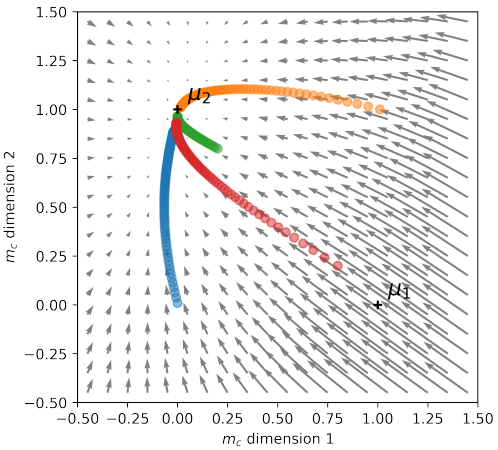
Memory retrieval: We have seen how the prior distribution $p(\boldsymbol{m}_c)$ could influence the dynamics of $\boldsymbol{m}_c$. For memory retrieval though, the dynamics of the hidden causes $\boldsymbol{m}_c$ should also be influenced by the bottom-up inference signal originating from the output prediction error. This influence is represented in figure 5.5, using the same PC-RNN-HC-A as before. By setting the parameters α_x and α_h to non-zero values, we activate the inference process for the hidden causes $\boldsymbol{m}_c$. At each time step, the error between the prediction $\boldsymbol{x}_t$ and the target $\boldsymbol{x}_t^*$ induces an update of $\boldsymbol{m}_c$ in a direction that decreases this error. The vector fields represented in figures 5.5a and 5.5b correspond to the average of the vector fields induced by the 60 data points forming the trajectories for the a and b sequence patterns.

Using only the inference signal and ignoring the influence of the prior distribution on $\boldsymbol{m}_c$, the dynamics of $\boldsymbol{m}_c$ are drawn towards the mixture density mean corresponding to the provided target pattern. We can observe that the vector field contains some areas with low gradients, with the dynamics of $\boldsymbol{m}_c$ converging very slowly to the attracting values, especially in figure 5.5a.

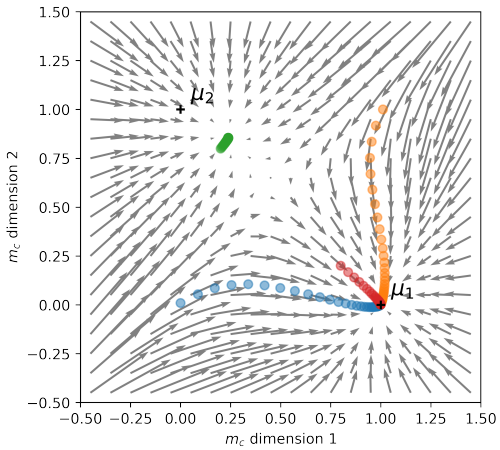
In figures 5.5c and 5.5d, we represent the dynamics induced by the combined influence of the prior probability distribution and the bottom-up inference mechanism. We obtain dynamics that converge faster to the correct prior density mean. Interestingly, we can see that the addition of the bottom-up prediction error signal is enough to deviate the blue (respectively the orange) trajectory



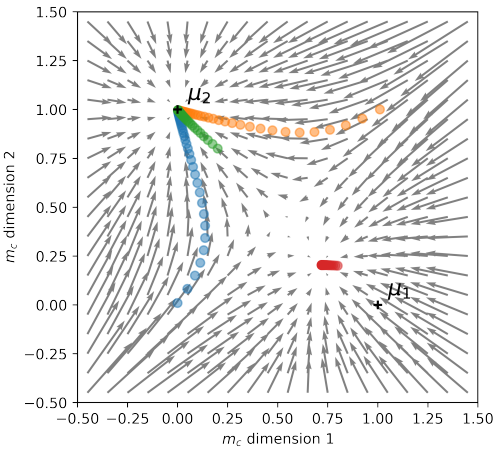
(a) Dynamics of $\boldsymbol{m}_c$ dictated by the bottom-up inference using a target corresponding to the first trained pattern.



(b) Dynamics of $\boldsymbol{m}_c$ dictated by the bottom-up inference using a target corresponding to the second trained pattern.



(c) Dynamics of $\boldsymbol{m}_c$ using both mechanisms using a target corresponding to the first trained pattern, with $\sigma_c = 0.4$.



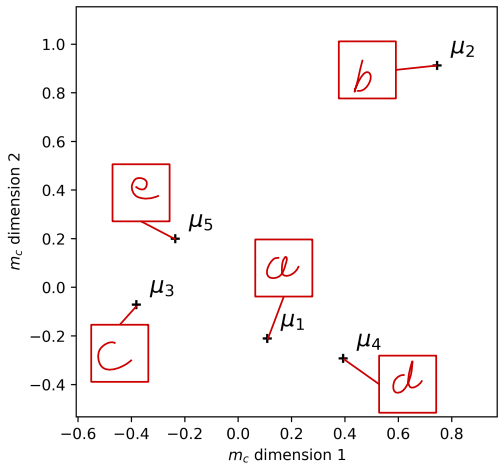
(d) Dynamics of $\boldsymbol{m}_c$ using both mechanisms using a target corresponding to the second trained pattern, with $\sigma_c = 0.4$.

Figure 5.5: Influence of the bottom-up inference on the dynamics of $\boldsymbol{m}_c$.

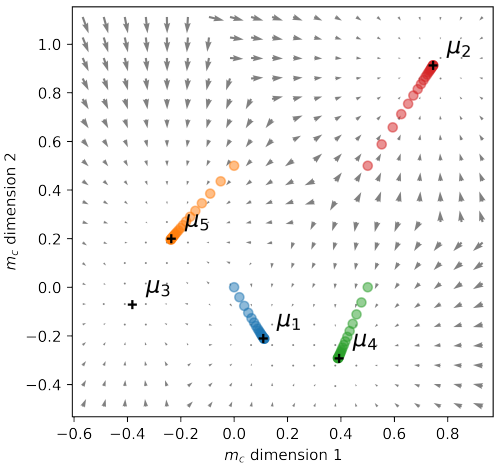
towards the prior mixture density mean $\boldsymbol{\mu}_1$ (respectively $\boldsymbol{\mu}_2$) even though the initial hidden causes were closer to $\boldsymbol{\mu}_2$ (respectively $\boldsymbol{\mu}_1$). However, we can see that the green trajectory in figure 5.5c and the red trajectory in figure 5.5d are stuck in local minima.

We propose to cast memory retrieval as a process of inference of the category z_k and the associated hidden causes vector $\boldsymbol{\mu}_k$ that can be used to regenerate a trajectory pattern $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$. In this view, memory retrieval is a dynamical process during which the hidden causes vector $\boldsymbol{m}_c$ evolves until it reaches the value $\boldsymbol{\mu}_k$ that properly regenerates the desired sequence. An issue with the described mechanism is that it may easily be stuck in a local minimum of the VFE, as both the inference mechanism, and the influence of the prior distribution, can induce local attractor basins.

Indeed, in sequence memories where a large number of patterns have been written, the dynamics of $\boldsymbol{m}_c$ dictated by the bottom-up inference mechanism alone are not as effective as the ones shown in figures 5.5a and 5.5b (which are obtain in a sandbag experimental set up with $n = p = d_c = 2$). On the other hand, if we consider the additional influence of the prior $p(\boldsymbol{m}_c)$ we can see that local minima can emerge in the optimization landscape as in figures 5.5c and 5.5d. We speculate that the additive noise influence can help mitigating this issue.



(a) Representation of the mixture density means $\boldsymbol{\mu}_k$ in the hidden causes space and their associated output sequence patterns.



(b) Dynamics of $\boldsymbol{m}_c$ dictated by the Gaussian mixture prior without any influence of the inference mechanism, with $\sigma_c = 0.1$.

Figure 5.6: Gaussian mixture prior after learning.

5.3.4 Structured case

In the previously presented method, we have assumed that the mixture density means $\boldsymbol{\mu}_k$ formed an orthonormal basis of the hidden causes space of dimension $d_c = n$. Consequently, the prior distribution $p(\boldsymbol{m}_c)$ induces dynamics where all the attraction basins are separated by the same distance. For memory retrieval, it might be more efficient to have prior density means corresponding to similar sequence patterns to be closer in the hidden causes space.

In this section, we propose to relax the constraint of freezing the values of $\boldsymbol{m}_c$ during training of the PC-RNN-HC-A model. Instead, we turn on the hidden state and hidden causes inference process during learning. After each training iteration on one sequence pattern k, we use the inferred value of $\boldsymbol{m}_c$ after presentation of the $T = 60$ targets x_t^* as the initial hidden causes value for the next iteration.

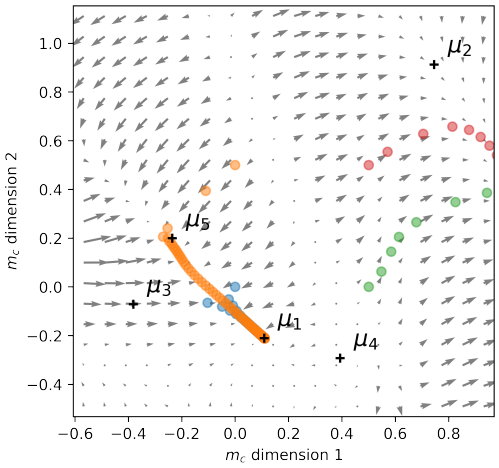
At the end of training, each Gaussian mixture mean $\boldsymbol{\mu}_k$ is initialized as the inferred hidden causes value for the sequence pattern k. Therefore, the Gaussian mixture means still correspond to the keys associated with the learned patterns in our sequence memory. If we use the RNN model strictly in prediction mode (by turning off the feedback pathway) using $\boldsymbol{\mu}_k$ as initial hidden causes, the RNN properly regenerates the k-th sequence pattern.

As a toy example, we propose to use a hidden causes dimension of $d_c = 2$. We train the model on 5 possible sequence patterns of the handwriting data set (the letters a, b, c, d, and e). The figure 5.6a displays the five trajectory patterns as well as the corresponding mixture density means $\boldsymbol{\mu}_k$ in the 2D hidden causes space. We represented in figure 5.6b the hidden causes dynamics induced by the Gaussian mixture prior on $\boldsymbol{m}_c$, with $\sigma_c = 0.1$. As in the last sections, we also display example trajectories starting from four initial hidden causes values.

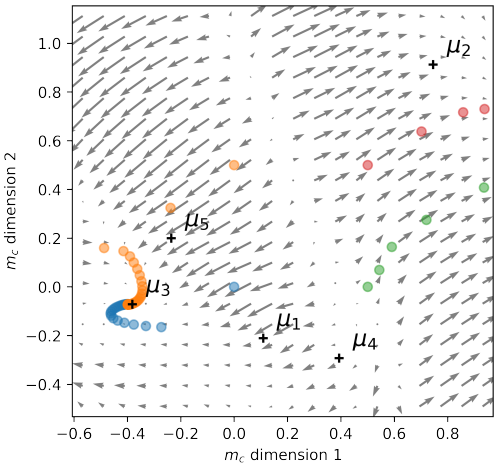
Again, we consider two tasks using this *structured* configuration:

Itinerancy: As in the previous section, we hypothesize that by alternating between phases characterized by low values of σ_c (for instance 0.1) and phases characterized by very high values of σ_c (such that the prior influence is neglected), $\boldsymbol{m}_c$ could exhibit itinerant dynamics that randomly transition between basins of attraction.

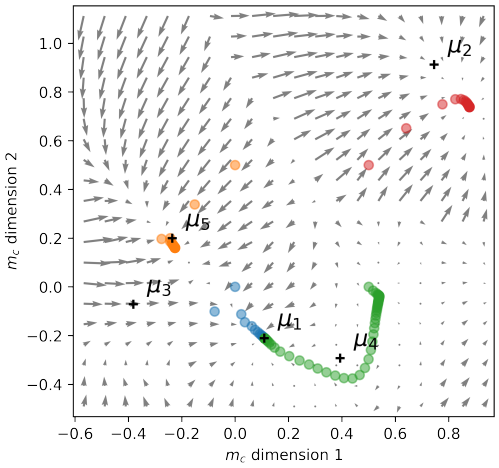
Memory retrieval: To be able to retrieve the hidden causes value $\boldsymbol{\mu}_k$ corresponding to a given pattern, we additionally consider the influence of the inference process on the hidden causes dynamics. In figures 5.7a and 5.7b, we display the hidden causes dynamics induced by the bottom-up inference process, using respectively the first sequence pattern a and the third sequence pattern c. Of the four displayed dynamics for $\boldsymbol{m}_c$, only two converge to the right mixture mean (respectively $\boldsymbol{\mu}_1$ and $\boldsymbol{\mu}_3$). In figures 5.7c and 5.7d, we display the hidden causes dynamics resulting by the combined influence the bottom-up inference process and the Gaussian mixture prior, using respectively the first sequence pattern a and the third sequence pattern c. We can see that adding



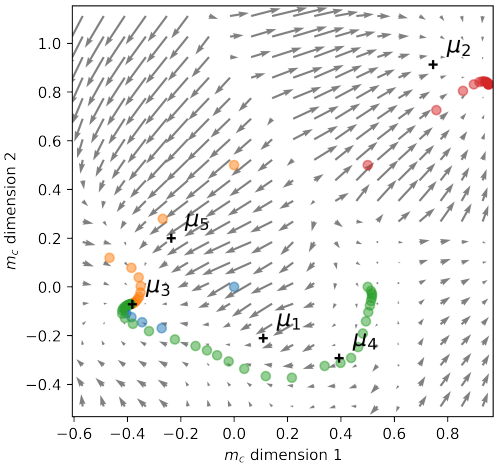
(a) Dynamics of $\boldsymbol{m}_c$ dictated by the bottom-up inference using a target corresponding to the first trained pattern (a).



(b) Dynamics of $\boldsymbol{m}_c$ dictated by the bottom-up inference using a target corresponding to the third trained pattern (c).



(c) Dynamics of $\boldsymbol{m}_c$ using both mechanisms using a target corresponding to the first trained pattern (a), with $\sigma_c = 0.1$.



(d) Dynamics of $\boldsymbol{m}_c$ using both mechanisms using a target corresponding to the second trained pattern (c), with $\sigma_c = 0.1$.

Figure 5.7: Influence of the inference mechanism onto the hidden causes dynamics, for the target sequence patterns a and c.

the prior influence does not prevent the dynamics of $\boldsymbol{m}_c$ to be stuck in local minima.

Again, we speculate that injecting additive noise in the hidden causes dynamics would help mitigating this issue. However, this noise might also have a negative impact on the memory retrieval process, by preventing the hidden causes dynamics to converge. Indeed, if the hidden causes vector $\boldsymbol{m_c}$ reached the correct key $\boldsymbol{\mu}_k^*$, the retrieval process should stop and not escape this basin of attraction because of the additive noise. To account for this problem, we suggest that at each time step t, the noise amplitude $\sigma_{r,t}$ is directly proportional to the prediction error on the hidden state layer. When the memory retrieval process is stuck in the wrong value, we expect this error to be significantly greater than when the memory retrieval process has reached the correct key $\boldsymbol{\mu}_k^*$. We propose to define the coefficient α_r such that:

$$\sigma_{r,t} = \alpha_r \|\boldsymbol{\epsilon}_{h,t}\|_1 \tag{5.13}$$

where $\|\cdot\|_1$ denotes the L1 norm. As such, α_r becomes the new hyperparameter controlling the general amplitude of the additive noise.

5.3.5 Summary of the proposed methods

Gaussian mixture prior means $\boldsymbol{\mu}_k$	Stages	Mechanisms	Parameters
Unstructured	Learning	No inference	
	Itinerancy	Random walk	σ_r
		Prior influence	α, σ_c
	Memory retrieval	Random walk	α_r
		Inference	α_x, α_h
		Prior influence	α, σ_c
Structured	Learning	Inference	$\alpha_x > 0, \alpha_h > 0$
	Itinerancy	Random walk	σ_r
		Prior influence	α, σ_c
	Memory retrieval	Random walk	α_r
		Inference	α_x, α_h
		Prior influence	α, σ_c

Table 5.1: Summary of the proposed methods.

Table 5.1 provides a summary of the proposed methods. The only difference between the structured and unstructured cases is that in the structured case we turn on the inference of the hidden causes during learning.

The dynamics of $\boldsymbol{m}_c$ can be influenced by three mechanisms. Whether we consider itinerant dynamics, or memory retrieval dynamics, we assume that the hidden causes $\boldsymbol{m}_c$ undergo a Gaussian random walk of variance σ_r^2, and that the hidden causes are influenced at each time step by the prior $p(\boldsymbol{m}_c)$ which shape is parameterized by σ_c. The difference between what we call itinerancy and memory retrieval is whether we consider the influence of the bottom-up prediction error signal for the inference of $\boldsymbol{m}_c$. In itinerancy, we do not assume that a target is provided and the network dynamics randomly transition between several hidden causes configurations. In memory retrieval, a target $\boldsymbol{x}_t^*$ is provided and the hidden causes dynamics are pulled in directions that minimize the prediction error at each time step.

5.4 Results

In this section, we present the results obtained using the itinerancy and memory retrieval methods we have presented. We conducted two experiments:

- Itinerancy using unstructured hidden causes values, using the PC-RNN-HC-M model.
- Memory retrieval using structured hidden causes values, using the PC-RNN-HC-A model.

5.4.1 Unstructured itinerancy

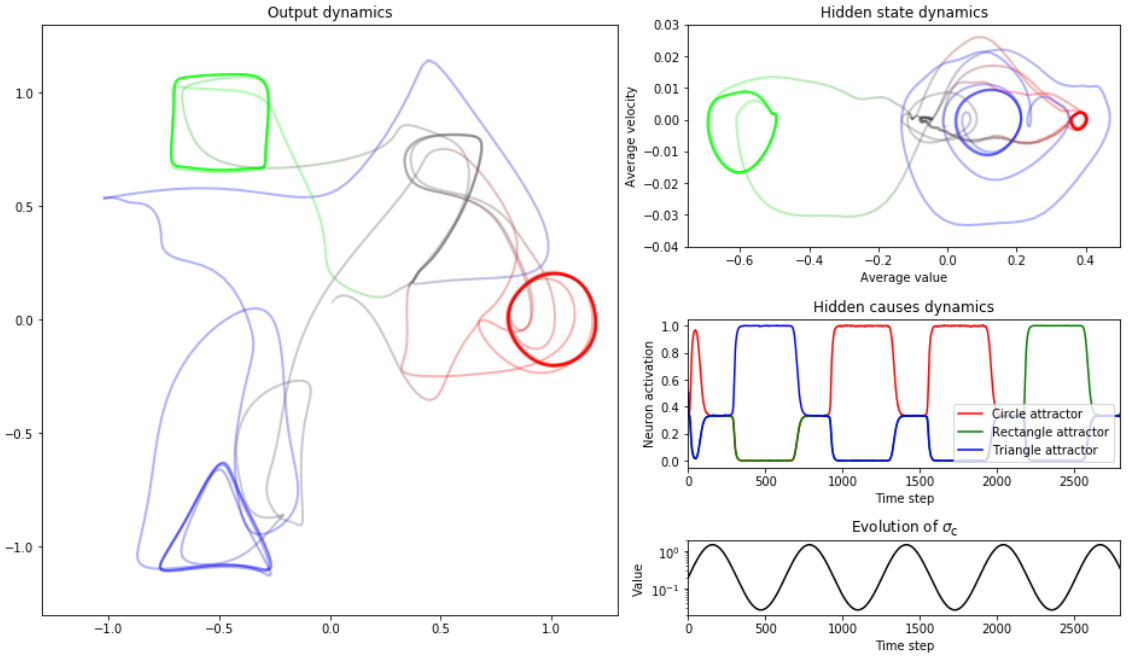


Figure 5.8: Simulation of itinerant dynamics in the unstructured PC-RNN-HC-M model. **Left:** Output trajectory generated by the model in mode A. The line colors in RGB values correspond to the activations of the three neurons of $\boldsymbol{c}$ throughout the trajectory. **Top-right:** Average velocity of the hidden state according to its average value throughout the trajectory. **Middle-right:** Evolution of the three hidden causes neuron activations over time. **Bottom-right:** Evolution of the σ_c coefficient over time.

We start by experimenting with itinerancy in the unstructured case. We first train the PC-RNN-HC-M model to generate three cyclic patterns for the three hidden causes values $\boldsymbol{\mu}_1 = (1, 0, 0)$, $\boldsymbol{\mu}_2 = (0, 1, 0)$ and $\boldsymbol{\mu}_3 = (0, 0, 1)$. The PC-RNN-HC-M model is dimensioned with $d_c = 3$, $d_h = 100$ and $d_x = 2$. The target cyclic patterns correspond respectively to a circle, a square and a triangle in 2D. The network is trained with BPTT using a learning rate $\lambda = 0.001$. After training, the PC-RNN-HC-M model has learned a different cycle attractor for the hidden state dynamics, for each value of $\boldsymbol{m}_c$, and learned the output weights transforming each attractor into the desired output pattern.

We implement the itinerancy method described previously, with σ_c varying according to the function $\sigma_c(t) = 0.2 \cdot \exp\{2\sin(t/100)\}$. This function periodically alternates between phases where σ_c is low, and phases where σ_c is high. The results are recorded in figure 5.8, and can be better visualized in this video. They were obtain using $\sigma_r = 0.01$ and $\alpha = 0.1$ as parameters. The source code for this experiment is available on GitHub[1].

We can observe that the RNN switches between the three attractors. When σ_c is high, the hidden causes converge towards the center value $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. This center value corresponds to the hidden state dynamics and output dynamics depicted in gray. This value of the hidden causes seems to correspond to a point attractor for the dynamics of $\boldsymbol{m}_h$, which was not something directly enforced by the training procedure. Starting from this configuration, when σ_c decreases, the hidden causes fall into one of the three attracting configurations that were trained to correspond to the three limit cycle attractors.

We want to verify whether the attractor switching behavior follows a uniform probability distribution or if some transitions are more likely to occur than others. We view the RNN as a

[1] https://github.com/sino7/random_itinerant_dynamics_in_pc_based_rnn

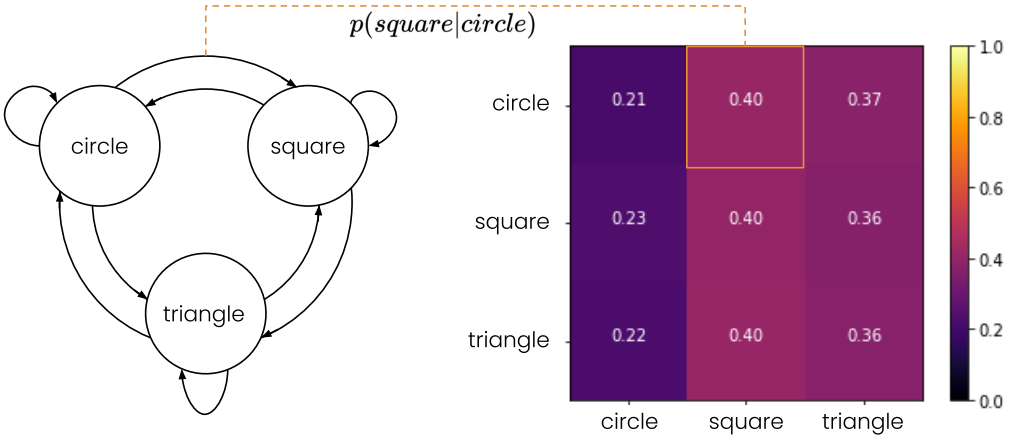


Figure 5.9: Markov chain and associated transition matrix for the itinerant dynamics in the unstructured PC-RNN-HC-M model. Lines in the matrix correspond to previous states and columns to next states. For instance, the estimated probability of switching from circle to square attractors is 0.40.

Markov chain with three configurations. We record 2000 attractor transitions that we use to build an estimation of the transition matrix of that Markov chain. The results are displayed in figure 5.9.

We can see that the probability of switching to a certain state seems independent from the previous state. This result can be explained by the fact that the intermediary, neutral configuration that the networks reaches before switching to a new configuration corresponds to a fixed point. If we let enough time for the hidden state to reach this fixed point, it would no longer hold any memory of the previous configuration.

We have shown how an RNN model implementing PC could exhibit attractor switching behaviors using an input noise signal. Here, we compare our results with other works aiming at modeling this behavior.

The approach described in (Yamashita and Tani, 2008) and (Namikawa et al., 2011) requires to train a separate RNN for each target sequence. In opposition, we have shown that our model can embed different dynamics within one RNN, and as such should scale better to an increased number of trajectories. On the other hand, one limitation of the model presented by (Inoue et al., 2020) is that quasi-attractors have a set duration, and the dynamics they yield cannot last longer than this trained duration before falling into a chaotic regime. In contrast, since our model relies on real trained limit-cycle attractors, any periodical trajectory can be maintained for as long as desired.

In order to draw a connection with CI, we have trained our PC-RNN-HC-M network to generate cyclic patterns. This possibility was not discussed previously, but those results also show that cyclic sequence patterns can be written in our sequence memory model.

5.4.2 Unstructured memory retrieval

Now, we depart from the itinerancy simulations and focus on the memory retrieval process. As explained in the methods section, for memory retrieval the dynamics of $\boldsymbol{m}_c$ are also influenced by the bottom-up inference process.

For the following experiment, we train a sequence memory of $p = 20$ 2-dimensional sequential patterns of length 60 using the PC-RNN-HC-A model. The memorized trajectories are taken from our handwriting data set, and thus correspond to sequences of (x, y) pen positions recorded during the handwriting of Latin alphabet letters. During the training of each trajectory pattern k ($1 \leq k \leq p$), the RNN hidden state is initialized using the same random-valued vector. The hidden causes are initialized randomly during the first presentation of the sequence pattern to learn. We turn on the inference of $\boldsymbol{m}_c$ during training, and use the value of $\boldsymbol{m}_c$ at the end of the trajectory as initial value for the next presentation of the sequence pattern. Basically, $\boldsymbol{m}_c$ is

seen as a model parameter trained during this learning process. For each trajectory $(x_1^*, \cdots, x_T^*)_k$, we learn corresponding hidden causes $\boldsymbol{m}_{c,k}$. During memory retrieval, these k values are used as Gaussian mixture means: $\boldsymbol{\mu}_k = \boldsymbol{m}_{c,k}$. This additional learning allows embedding the hidden causes in a space of arbitrary dimension, and to capture regularities among the trajectories to learn. We expect similar trajectories to have similar corresponding hidden causes after learning. Hidden causes can be seen as learned representations of the trajectories.

During memory retrieval, we initialize the hidden causes without any prior knowledge of the target pattern. The goal of the process in precisely to retrieve the hidden causes corresponding best to the target trajectory. The joint influence of the bottom-up prediction error signal and top-down prior distribution, together with additive noise, yield memory retrieval trajectories than can be recorded as sequences of values of $\boldsymbol{m}_c$. After each trial (i.e. a presentation of the full target sequence), we record the inferred hidden causes value. The memory retrieval process stops when the hidden causes reach a value associated with a low VFE evaluation. To achieve a low VFE, the hidden causes value must be probable under the prior distribution, and must properly predict the target sequence. We define a VFE threshold and record the number of trials necessary before reaching a configuration that falls under this threshold. We call this number of trials the *retrieval time*.

We conduct an hyperparameter search on the hidden causes dimension d_c, the prior distribution standard deviation σ_c, the update rates α and α_h, as well as on the coefficient α_r responsible for the amplitude of the additive noise. We use the hyperparameter search method that we described in the previous chapter. The scoring function for this hyperparameter search is the average retrieval time as defined in the previous paragraph. For each possible hyperparameter configuration, we train our PC-RNN-HC model for 1000 iterations using BPTT with a learning rate of 0.03, on the 20 trajectories of the handwriting data set. After training, we run a memory retrieval process starting from $\boldsymbol{m}_c = \boldsymbol{0}$ for each of the $p = 20$ target trajectories. The average retrieval time on these 20 retrievals is used to assess the quality of the hyperparameter configuration. If the memory retrieval has not converged within the 1000 trials, we set the retrieval time to 2000 (as a way to penalize these situations). The hyperparameter search finds the following optimal values for the hyperparameter being optimized:

- $d_c = 2$
- $\sigma_c = 9.6 \times 10^{-2}$
- $\alpha = 8.5 \times 10^{-3}$
- $\alpha_h = 1.9 \times 10^{-2}$
- $\alpha_r = 2.6$

Additional parameters of the PC-RNN-HC-A are set manually, using the following values:

- $d_h = 50$
- $\tau_h = 50$
- $\alpha_x = 0.1$

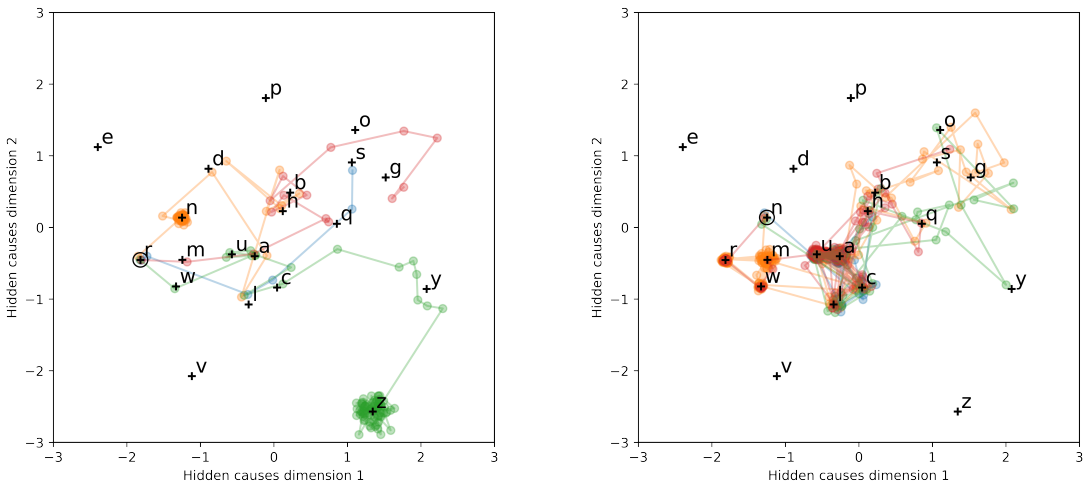
The source code containing the implementation of the memory retrieval algorithm is available on GitHub[2].

5.4.2.1 Memory retrieval dynamics

Since the optimal hidden causes dimension is $d_c = 2$, after training, we can represent in a 2D space the learned hidden causes representation for each of the target trajectory in the handwriting data set.

Figure 5.10 displays these representations, as well as four memory retrieval trajectories for both target patterns r and n. Each colored dot represents the value of the hidden causes at the end of one complete trial of 60 time steps. To obtain trajectories that were easier to visually interpret, we have slowed down by a factor of 2 the dynamics of the hidden causes variable, only for this figure.

[2]https://github.com/sino7/predictive_coding_for_memory_retrieval



(a) Memory retrieval with the target pattern r.

(b) Memory retrieval with the target pattern n.

Figure 5.10: Hidden causes trajectories during memory retrieval for different target patterns using the structured PC-RNN-HC-A model. The Gaussian mixture mean corresponding to the target pattern is circled in black. In each figures, four trajectories starting from random initial hidden causes are shown. Each trajectory is shown in a different color (blue, orange, green, red).

This was done by dividing by 2 the coefficients α_h and α, and α_r. Consequently, if a memory retrieval trajectory seems to last for 20 trials in this figure, in practice it could last for 10 trials.

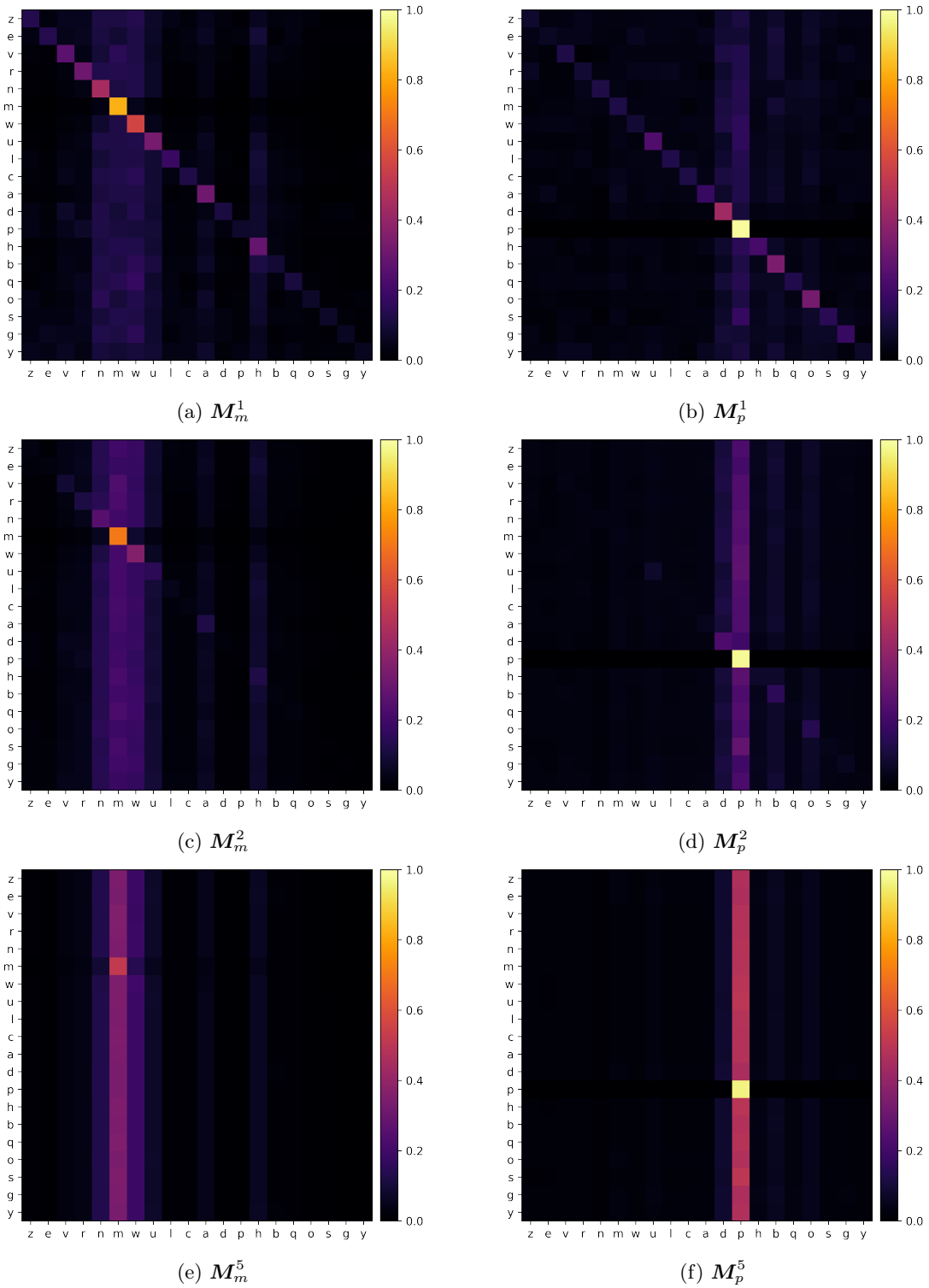
First, we can see that the learned representations carry some meaningful structure. Letter sequential patterns with similar motion seem to have similar hidden causes representation. This can be seen for instance with the representations of the letters b and h being very close. Both these letters are written by first moving the pen down, and then drawing a loop on the right, this loop being incomplete for the h. Over many different trained models, we always observe this pair of letters in the hidden causes space.

Using the target trajectories corresponding to the letters r and n, we have obtained memory retrieval trajectories that always converge to the correct Gaussian mixture mean. Figure 5.10a shows memory retrieval trajectories that converged in less than 50 trials, except for the one represented in green, that started near the attracting mixture mean $\boldsymbol{\mu}_z$, and difficultly managed to escape this basin. Figure 5.10b shows particularly bad memory retrieval trajectories. We can see that the red and orange trajectories are oscillating between the mixture means $\boldsymbol{\mu}_r$, $\boldsymbol{\mu}_m$ and $\boldsymbol{\mu}_u$ for a long time before reaching $\boldsymbol{\mu}_n$. Still, in most cases, the memory retrieval converges to the correct mixture mean in less than 13 trials (the median retrieval time is 12).

We can estimate transition matrices for the memory retrieval iterative process. In this case, the memory retrieval iterative process can transition between attractor basins at each new trial. For this reason, we can identify at each new trial which transition was performed by comparing the current closest mixture mean with the closest mixture mean at the last trial. We count such transitions on 1000 trials for 100 different trained models, and accordingly estimate the transition matrices for each target pattern.

Figure 5.11 displays some of the obtained transition matrices. For the target patterns m and p, we have estimated the transition matrices $\boldsymbol{M}_m$ and $\boldsymbol{M}_p$. We also display these matrices at the power of 2 and 5, to show an approximation of the transition probabilities after respectively 2 and 5 trials. On the left, we can see that for the target pattern m, the retrieval mechanism is attracted towards the means $\boldsymbol{\mu}_n$, $\boldsymbol{\mu}_m$ and $\boldsymbol{\mu}_w$ almost indistinctively. After 5 transitions, we can see that the trajectory should have converged to one of these three attractors. On the right, we can see a better situation with the target pattern p, where only the correct mixture mean $\boldsymbol{\mu}_p$ seems to be attracting the memory retrieval trajectories.

We can conclude from these results that the iterative process we have proposed can be used to properly retrieve patterns written inside our sequence memory.



(a) $\boldsymbol{M}_m^1$ (b) $\boldsymbol{M}_p^1$

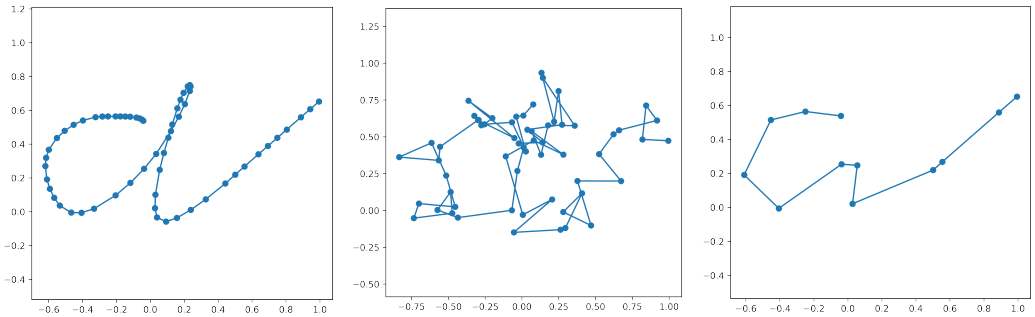
(c) $\boldsymbol{M}_m^2$ (d) $\boldsymbol{M}_p^2$

(e) $\boldsymbol{M}_m^5$ (f) $\boldsymbol{M}_p^5$

Figure 5.11: Transition matrices for 1, 2 and 5 transitions for the target patterns m and p in the structured case for 20 temporal patterns, obtained by simulating 20 memory retrievals on 500 different trained models. The list of temporal patterns has been permuted to highlight the formation of some clusters. For instance, the patterns corresponding to the letters *v, r, n, m, w, u* share some similarities that are reflected in the transition matrices.

5.4.2.2 Memory retrieval using approximate targets

In the previous experiment, the target pattern provided for memory retrieval always exactly matched one of the learned patterns in the sequence memory. It would also be interesting to assess the performance of this retrieval method when the target sequence comprises errors, or is incomplete. The goal of this memory retrieval process is indeed to be able to recover a memory item starting from an approximate version of this item. In this subsection, we conduct two additional experiments using this model. In the first experiment, we apply an additive noise on the target trajectory. In the second experiment, we mask a part of the trajectory.



(a) (b) (c)

Figure 5.12: Illustration of the effects of additive noise or masking onto the target trajectories. Each dot represents one point of the sequence. The three sequences correspond to the patter a. On the second trajectory we have applied an additive noise of standard deviation $\sigma_{noise} = 0.1$. On the third trajectory we have masked 80% of the points.

Figure 5.12 shows the effect of these two modifications onto the information available for memory retrieval. The correct trajectory for the sequence pattern a is shown on the left. In the middle is shown the same trajectory with an additive noise of standard deviation $\sigma_{noise} = 0.1$ on each point. On the right is shown the same trajectory with only 12 points over the 60 points initially composing the trajectory.

First, we study the effect of an additive noise applied to the target trajectory. For each point of the target trajectory, we add a random 2D value sampled from a bivariate Gaussian of mean 0 and covariance $\sigma^2_{noise}\mathbb{I}_2$. We vary the parameter σ_{noise} and measure how this influences the retrieval time using our method.

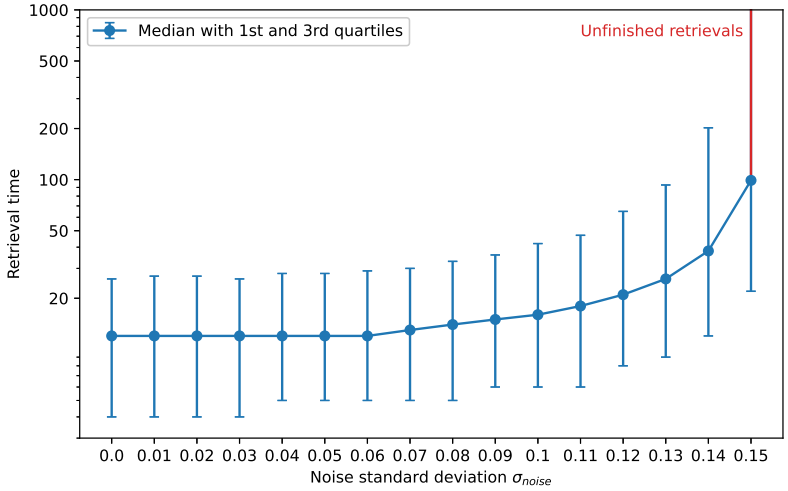


Figure 5.13: Distribution of the memory retrieval time according to the noise standard deviation σ_{noise}.

Figure 5.13 displays the distribution of the retrieval time according to the noise standard deviation σ_{noise}. Since we limit the duration of the retrieval process to 200 trials, there are some memory retrieval attempts that never converge. As such, evaluating the model performance according to the average retrieval time is impossible. For this reason, we instead measure the median retrieval time, as well as the first and third quartiles of the retrieval time distributions. This figure represents the evolution of these quantities according to σ_{noise}. The medians, first quartiles and third quartiles are displayed on a logarithmic scale. We can see that the retrieval mechanism is not significantly impaired by the noise for values $\sigma_{noise} \leq 0.1$. However, when further increasing the noise amplitude, we can see that the proportion of unfinished retrieval rapidly increases. We believe that this is due to the VFE threshold that we have defined. Reaching this threshold indicates that the agent has *recognized* the provided target pattern. Applying too much noise onto the target prevents this method from converging. We could artificially increase the VFE threshold, but in this case the memory retrieval might accept hidden causes configurations that do not actually correspond to a prior mixture mean $\boldsymbol{\mu}_k$, which we want to avoid. Still, we can conclude that the memory retrieval process that we have proposed can retrieve the correct pattern even with approximate versions of the actual memory item.

We now turn to the second experiment. This time, the points composing the target sequence pattern are not modified, but only a proportion of those are provided to the agent for memory retrieval. To implement this, we simply set to 0 the prediction error obtained with data points that we want to be hidden. This way, the PC-RNN-HC-A model cannot use this information to update the hidden causes $\boldsymbol{m}_c$. We note the proportion of hidden data points p_{mask}, and study the influence of this value onto the retrieval time.

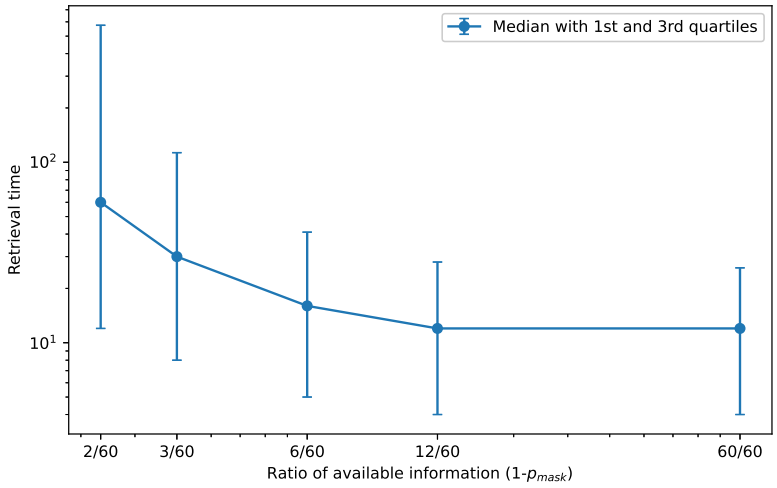


Figure 5.14: Distribution of the memory retrieval time according to the mask probabilities p_{mask}.

Figure 5.14 displays the distribution of the retrieval time several values of p_{mask}. We can see that masking 90% of the information does not seem to significantly impact the retrieval time. To better understand how much information is available to the agent, we can count the number of data points actually used for memory retrieval. We can see in this figure that the memory retrieval process can still converge most of the time using only two points among the 60 points composing the trajectories.

5.4.2.3 Scaling memory retrieval

Finally, we can question whether this memory retrieval method can scale to larger sequence memories. Using $p = 20$ sequence patterns of the handwriting data set, we obtained a median retrieval time of $T_r = 12$. Using a more simple program iterating over all possible keys, the memory retrieval would need between 1 (in the best scenario) and p (in the worst scenario) trials. Even though the method we have presented can benefit from a learned structure of the hidden causes representations of the memory items, it does not outperform this simple search algorithm.

Still, it is possible that the proposed method scales better to large sequence memories. Consequently, we perform an additional experiment where we learn larger sequence memories on the simple data set presented in the previous chapter. We train sequence memories of size 15, 30, 60, 120 and 240. For each sequence memory, we ensure that the number of model parameters (using the PC-RNN-HC-A) is proportional to p, and run an hyperparameter search to find the best hyperparameters for memory retrieval. We also authorize larger sequence memories to train for a longer time.

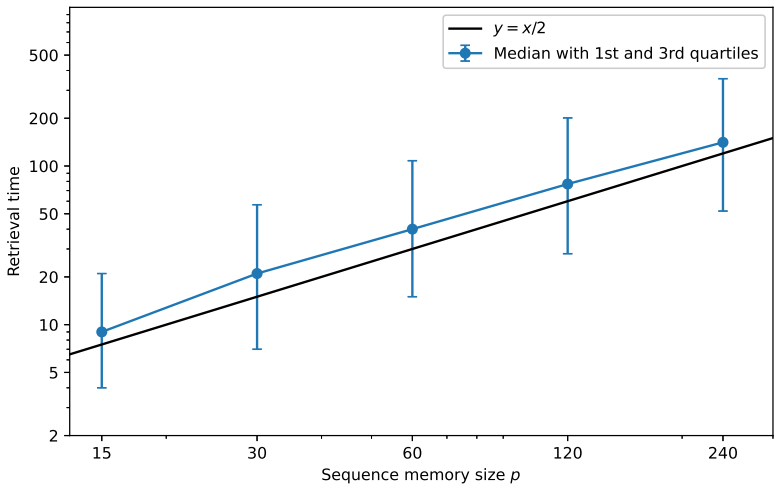


Figure 5.15

Figure 5.16: Distribution of the memory retrieval time according to the number of temporal patterns p in the sequence memory.

Figure 5.16 displays the distribution of the retrieval time for several sequence memory sizes. We also display in black the median retrieval time that we would obtain with a simple search algorithm that scales linearly with p.

We can see that our memory retrieval method seems to scale linearly with the size of the sequence memory. This suggests that the learned structure does not properly guide the retrieval mechanism in the same way that sorting a list makes it possible to have search algorithms that scale logarithmically with the list length. The failure of our memory retrieval algorithm to scale sublinearly could be the result of several things. For instance, the large number of patterns in memory could negatively affect the gradient descent on the prediction error. By increasing the number of patterns, the dynamics induced by the bottom-up inference might become more imprecise at guiding $\boldsymbol{m}_c$ towards the correct value. This would make the influence of the bottom-up prediction error signal almost useless at retrieving the correct key, transforming our memory retrieval algorithm as a naive random walk with no preferred direction.

5.5 Discussion

Compared to the models we have derived in the last chapter, this model involves more complex computations. In particular, the hidden causes update involves the computation of the true posterior probability $p(Z|\boldsymbol{m}_{c,t})$ at each time step t. Our model might scale better to very large sequence memories if we approximate this mechanism. The role of this mechanism being to pull $\boldsymbol{m}_c$ towards preferred values $\boldsymbol{\mu}_k$, it could for instance be replaced with a Hopfield network with states of low energy corresponding to the values $\boldsymbol{\mu}_k$.

A weakness of the method we have presented is that the retrieval time seems to scale linearly with the number of patterns written in the sequence memory. We speculate that this issue comes from the inefficiency of the gradient descent on VFE that takes place in our models. These difficulties could be due to natural limitations of PC. The derivations of the PC models assume

that the recognition densities $q(\boldsymbol{H})$ and $q(\boldsymbol{C})$ are unimodal and can be parameterized using $\boldsymbol{m}_h$ and $\boldsymbol{m}_c$. For our memory retrieval task, it could be more efficient to have a multimodal recognition density $q(\boldsymbol{C})$ that jointly considers several possible values for $\boldsymbol{C}$.

The inefficiency of the inference mechanism in PC could also more simply come from the fact that both inference and generation are supported by the same RNN model. In comparison, direct classification of sequences (or sequence representation) can be performed using another RNN instance. We have imposed ourselves with the constraint that the encoding (inference) and decoding (generation) should be performed on a unique model. Relaxing this constraint, we could use one RNN for the encoding $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*) \rightarrow \boldsymbol{c}$, and another one for the decoding $\boldsymbol{c} \rightarrow (\boldsymbol{x}_1, \cdots, \boldsymbol{x}_T)$. This type of architectures has been widely used for sequence-to-sequence (seq2seq) tasks (Sutskever et al., 2014), and also includes autoencoders.

Finally, the encountered difficulties might come from the dynamic nature of the RNN models. If we were to consider feedforward neural architectures able to generate temporal patterns, such as the CNN model presented in chapter 3, the inference mechanisms provided by PC might be more efficient than with RNN-based generative models.

In summary, we believe that these limitations come from the inefficiency of the PC-based inference method in RNNs. Directions of improvement could be to use feedforward models together with PC, or to use another variational inference method, for instance relying on autoencoders.

Although our performances are not impressive, we have still shown that it is theoretically possible to design robust reading mechanism for a long-term memory of temporal patterns using PC-based RNN models. This reading mechanism is able to retrieve the correct memory item even in the presence of noise or with a large proportion of missing information in the provided sequence.

Chapter 6

Motor trajectories learning

6.1 Introduction

In the previous chapters, we have studied most of the questions that we have raised about sequence memories. We have proposed several RNN architectures based on the PC theory and investigated their memory capacity, as well as the behaviors of our learning algorithms in a continual learning setting. In the last chapter, we have developed iterative methods for memory retrieval by iterating on these RNN models.

The question that we have left untouched until now is the problem of indirect supervision for sequence memory learning. Indeed, in all our previous experiments, we have considered that the temporal patterns $(\boldsymbol{x}_1^*, \cdots, \boldsymbol{x}_T^*)$ to write into the sequence memory (through learning) were directly available as target observations that the generative memory tries to predict. If we could additionally endow our models with the ability to construct adequate temporal patterns without relying on this direct supervision, the resulting sequence memory models could for instance be used to store motor trajectories with no direct supervision in the motor space. In this chapter, we approach this question from the perspective of Active Inference (AIF), a mathematical framework embedded in the FEP theory that frames action (or motor control) as another process driven by the minimization of VFE (or related quantities, as we will see). Since this framework is based on the same theoretical ground, it aligns nicely with the models we have derived so far. Building upon our previous work, we design several models for motor sequence memory learning applied to a handwriting task for a robotic arm. We show that learning methods based on AIF and PC can be used to build a robust motor sequence memory of handwritten patterns (for instance, letters) using a visual teaching signal.

The chapter is organized as follows. In section 6.2, we provide a literature review of the AIF theoretical framework. In section 6.3, we build upon previously existing methods to design a principled framework for motor sequence memory learning using AIF. Sections 6.4 and 6.5 report two published works attempting to implement this proposed framework respectively in an unsupervised learning setting and an imitation learning setting. Finally, we conclude this chapter in 6.6 by discussing our findings, the possible improvements of our models, and open questions.

6.2 Related work

Artificial and biological agents have to learn how to choose among different possible courses of action in an adaptive manner. Decisions can be driven by different factors, that are usually classified into two categories: extrinsic (or pragmatic, instrumental) and intrinsic (information-seeking, epistemic) rewards. These different drives for decision-making can be composed in many ways, and actions can be motivated by different drives at the same.

The question of decision-making has been studied in depth in the field of Reinforcement Learning (RL) with great successes in the last decades, for instance in playing video games (Mnih et al., 2013; Berner et al., 2019) and board games (Silver et al., 2016; Silver et al., 2017; Schrittwieser et al., 2020), robot control (Nagabandi et al., 2020), visual navigation (Zhu et al., 2017; Mirowski et al., 2016). In this framework, an agent is driven by the maximization of its expected return, which is defined as the expected sum of (possibly discounted) rewards over time. Many different

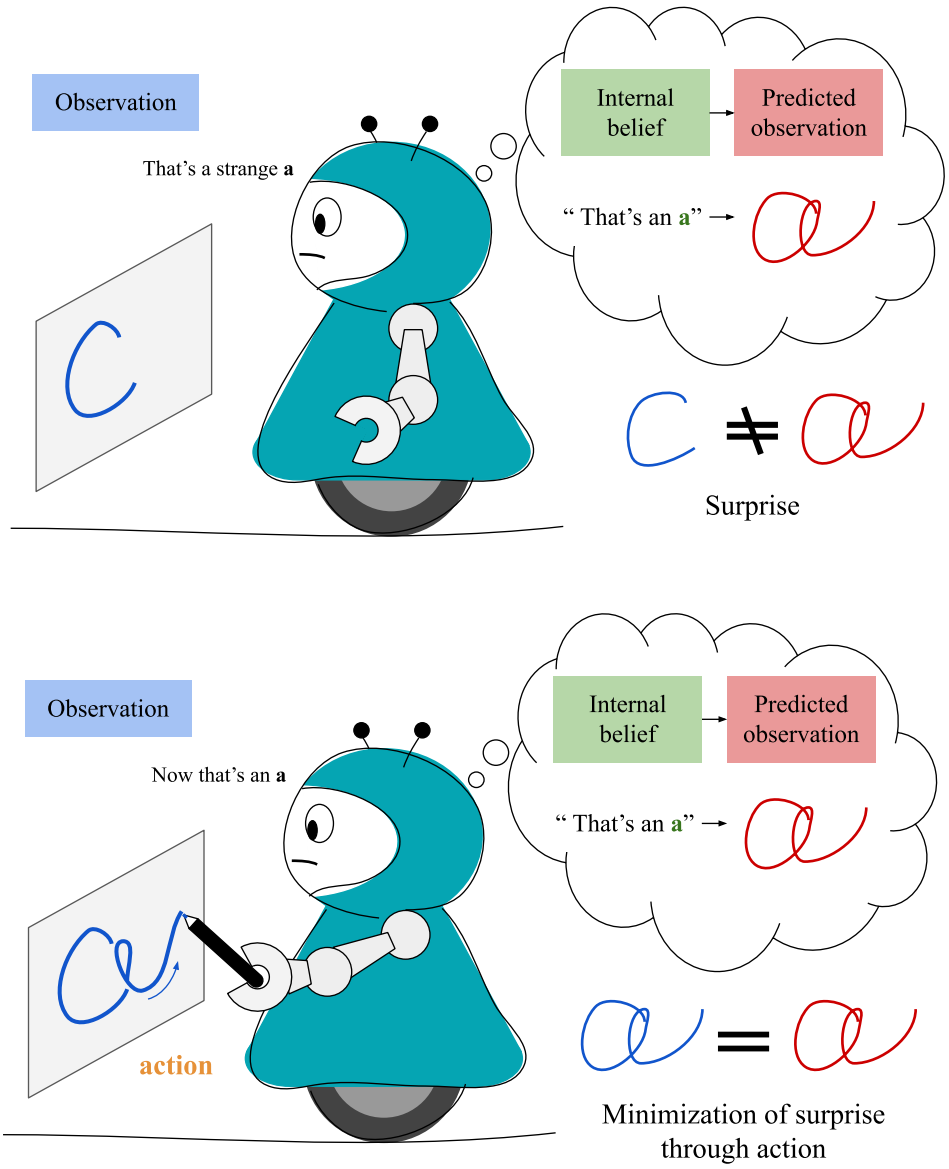


Figure 6.1: Illustration of surprise minimization through action.

methods have been proposed in RL to address this question, that we could classify according to several criteria: whether the agent learns an estimation of the expected return at each state (i.e. a value function), whether the agent exploits a learned model of the environment, whether policies are trained while being used by the learning agent, whether they assume that the environment is fully observable, etc.

In contrast, the Active Inference Framework (AIF) has been introduced in the FEP literature as a different approach to the question of learning adaptive behavior. This framework suggests that an agent acts in order to maximize evidence for a generative model naturally biased towards its preferences. To draw a parallel with RL, sensory observations that are likely under the agent's model can be considered more rewarding, and inversely, sensory observations that are surprising for the agent are seen as less rewarding. Figure 6.1 illustrates how an agent could minimize this surprise through action. The agent acts in order to fulfill its initially erroneous prediction. Because the agent's sensory prediction depicts an a, it actively tries to make this prediction come true, here by transforming the c into an a. This process minimizes surprise, since after acting on its environment the agent's prediction matches its observation.

6.2.1 Initial formulation of AIF

In early formulations of AIF (Friston and Kilner, 2006; Friston et al., 2009; Friston et al., 2011), actions are directly inferred through immediate minimization of VFE. The agent's generative model predicts proprioceptive sensory observations. VFE in this case can be minimized through perceptual inference, for instance using the PC mechanisms we have introduced earlier, but can also be minimized through action. Indeed, another way to minimize VFE is for the agent to act on its body in order to fulfill its proprioceptive prediction. This type of inference of motor commands has been related to classical reflex arcs, where reflexes are the immediate reactive actions to unpredicted proprioceptive states.

This formulation of AIF has been applied in several works. Recent implementations of this principle are presented in (Oliver et al., 2019; Lanillos, Cheng, et al., 2020). They use VFE as an objective function for the control of a humanoid robot for reaching and visual tracking. Dynamics on the internal state $\boldsymbol{h}$ are enforced by defining attractors $\boldsymbol{o}^*$ in the sensory space that should be reached. This is done using joint positions as internal states, and an inverse model to determine the internal states corresponding to desired target positions. By trying to fulfill the predictions of this biased generative model, the robot achieves the desired motor trajectories, with more accuracy than using inverse kinematics models. A similar scheme is proposed in (Pezzato et al., 2020) on a 7-DOF robot arm. The joint angle positions of the arm are again used as internal states, however, the dynamics of the internal states are enforced to reach target configuration $\boldsymbol{h}^*$ directly in the internal state space,and thus, in turn, in the joint angle position space. They show that using AIF with an approximate forward model and second-order generalized coordinates provides superior performances and lower computational cost than a state-of-the-art controller used for reference. Enforcing the internal state's trajectory to converge towards desired configurations $\boldsymbol{h}^*$ can actually be framed as having hidden causes corresponding to this target configuration $\boldsymbol{h}^*$ directly influencing the dynamics of the internal state. This formulation was for instance used in one of the first implementations of AIF for robot control (Pio-Lopez et al., 2016). In contrast, (Meo and Lanillos, 2021) do not use proprioceptive information as latent states, but instead, learn a multimodal VAE to model the relation between latent state and visual and proprioceptive information. Still, behavior is driven by desired proprioceptive states and not goals defined in another modality (here, visual), which facilitates the control process. For a recent and more complete review of existing implementations, we refer to (Ciria et al., 2021).

These approaches are limited by the fact that the agent behavior is primarily dictated by a biased generative model directly predicting what joint angle positions should be. The AIF framework provides reactive mechanisms to achieve this prediction, but cannot be used without having goals directly provided as desired joint angle positions. For agents evolving in complex realistic environments, it seems unlikely that behavior is entirely prescribed through generative models predicting proprioceptive observations, innately biased towards successful behavior. Instead, goals are more likely to be defined using a biased generative model on other modalities such as exteroceptive (e.g. low probability of smelling or tasting something that smells or tastes bad) or interoceptive (e.g. low probability of feeling hunger) observations. In such a situation, since the supervision provided by the biased generative model does not directly involve proprioceptive observations, the agent would need to understand how its actions affect exteroceptive or interoceptive observations to properly minimize VFE. To achieve this, internal models relating actions to their sensory consequences would need to be learned. An instance of the described approach is presented (Sancaktar et al., 2020). In this work, the authors control a robotic arm using VFE minimization based on visually defined goals. They use CNNs in order to predict the visual consequences of actions, and actions are inferred through BP of the prediction error through the CNN. In fact, as we will see shortly, BP based approaches extend very naturally to the more complex situation dealing with temporal actions and goals.

6.2.2 Expected free-energy

Other, more recent formulations of AIF (Friston et al., 2015; Friston et al., 2016) account for temporally extended actions, i.e. action sequences, sometimes also called policies (although not exactly equivalent to policies in RL). Since the VFE expression depends on the current observation, choosing a course of action that minimizes future VFE would require knowledge of future observations. Consequently, instead of minimizing the VFE objective function, these methods estimate

a quantity called Expected free-energy (EFE), corresponding to a cumulative sum of future VFE estimations. The sequence of actions is then optimized in order to minimize this quantity. If we denote by $\pi = (\boldsymbol{m}_1, \dots, \boldsymbol{m}_T)$ the sequence of actions (or motor commands), $\boldsymbol{o}_t$ the observation at time t, and $\boldsymbol{h}_t$ the hidden state of the generative model at time t, we can express the EFE at time step $t = \tau$ as:

$$EFE_\tau(\pi) = \underbrace{-\mathbb{E}_{q(\boldsymbol{o}_\tau, \boldsymbol{h}_\tau|\pi)}[\log p(\boldsymbol{o}_\tau)]}_{\text{Instrumental value}} \underbrace{-\mathbb{E}_{q(\boldsymbol{o}_\tau)}\big[D_{KL}[q(\boldsymbol{h}_\tau|\boldsymbol{o}_\tau, \pi)||q(\boldsymbol{h}_\tau|\pi)]\big]}_{\text{Epistemic value}} \tag{6.1}$$

where $p(\boldsymbol{o}_\tau)$ denotes the prior probability distribution on observations directly encoding the agent's preferences, and $q(\boldsymbol{o}_\tau, \boldsymbol{h}_\tau|\pi)$ denotes the recognition probability distribution on future observations and states, that implement the agent's predictions knowing the policy π. This expression is composed of two negative terms, and as such minimizing EFE corresponds to maximizing both terms. The first term, the instrumental value, measures how likely the expected observation $\boldsymbol{o}_\tau$ under q is according to the prior preferences p. This value is maximized when the policy π is associated with expected observations that are preferred according to the biased generative model p. This instrumental value can be directly related to the external reward used in RL.

The epistemic value measures the expectation of information gain according to the observation $\boldsymbol{o}_\tau$. Maximizing this term would encourage the agent to perform actions that lead to observations bringing new information to the agent about its latent state $\boldsymbol{h}_\tau$. Many different formulations of intrinsic motivation have been proposed (Oudeyer and Kaplan, 2009; Barto et al., 2013), suggesting to drive actions for instance by the research of surprise (Sun et al., 2011), novelty (Houthooft et al., 2016), empowerment (Klyubin et al., 2005), diversity of effects (Daucé, 2020), reducible prediction errors (Schillaci et al., 2020), or competitive self-play (Baker et al., 2019). An advantage of EFE is that it naturally combines an intrinsic (epistemic) value and an extrinsic (instrumental) value, which has the benefit of evaluating reward-seeking (exploitation) and epistemic foraging (exploration) behaviors using the same information-theoretic currency. Although it makes it possible to evaluate intrinsic and extrinsic rewards using the same measurement unit, it does not provide guidelines on how to balance these two drives.

While there are some works in the literature implementing planning with AIF, it has mostly been applied to toy examples in discrete state spaces (Friston et al., 2015; Friston et al., 2021; Tschantz et al., 2020b). Recently, some approaches have tried using deep neural networks as function approximators for the generative and recognition models in order to scale AIF to more complex problems (Millidge, 2020; Tschantz et al., 2020a; Çatal et al., 2020; Çatal et al., 2021). In this line of research, the work presented in (Çatal et al., 2021) is the first (to our knowledge) to apply AIF on a complex robotic task requiring perceptual inference (using VFE) for mapping and localization, as well as action planning (using EFE) for navigation.

6.2.3 Current discussions

Since AIF is a recent field of research, the theory and its implications are still being debated in the literature. Here we shortly present two points of discussion regarding the EFE formulation of AIF, and the relevance of AIF as a theory of adaptive behavior.

Coming back to the EFE formulation expressed in equation 6.1, it could come as a surprise that minimizing an estimation of future VFE would result in an objective function that directly preconizes the research of information gain. Initially, the VFE was derived as an upper bound on surprise, and only used as a proxy measure that could be minimized more efficiently. If AIF frames action as minimization of future surprise, then we would expect this scheme to encourage surprise avoidance over novelty search. Motivated by this apparent contradiction, some authors argue that the EFE might not be the natural extension of the VFE, and instead propose other objective functions more closely related to the minimization of surprise. For instance, (Millidge et al., 2021) formulate a novel objective function labeled Free-Energy of the Future (FEF) that can be derived more naturally starting from the expression of VFE.

$$FEF_\tau(\pi) = \underbrace{-\mathbb{E}_{q(\boldsymbol{o}_\tau, \boldsymbol{h}_\tau|\pi)}[\log p(\boldsymbol{o}_\tau|\boldsymbol{h}_\tau)]}_{\text{Accuracy}} + \underbrace{\mathbb{E}_{q(\boldsymbol{o}_\tau)}\big[D_{KL}[q(\boldsymbol{h}_\tau|\boldsymbol{o}_\tau, \pi)||q(\boldsymbol{h}_\tau|\pi)]\big]}_{\text{Complexity}} \tag{6.2}$$

Contrary to the EFE objective function, minimizing FEF encourages behaviors towards observations that keep $q(\boldsymbol{h}_\tau|\boldsymbol{o}_\tau, \pi)$ close to $q(\boldsymbol{h}_\tau|\pi)$, and therefore penalizes novelty. This decomposition

of FEF into two terms called accuracy and complexity is directly analogous to the decomposition of VFE, and it can be proved that this quantity actually provides an upper bound on expected future surprise. Even though this objective function might at first look like it would never encourage exploration, it might still be the case. Indeed, minimizing surprise should induce an agent to seek out states likely under its generative model, as well as states that would make the generative model (through perceptual inference and learning) able to predict the future states more properly. Since an agent minimizes an expected (possibly discounted) sum of future values, policies with a tendency to explore could be chosen if this translates in better exploitation (accuracy or instrumental value) in the long term.

There has been some criticism in psychology about casting surprise as a fundamental drive for action. One famous argument against AIF is known as *the dark room problem* (Friston et al., 2012; Sun and Firestone, 2020). Basically, it states that if the end goal pursued by biological agents was to minimize surprise, animals and humans would seek out regions of their state space (or environment) where there is no variation in sensory observations, such as a dark room. As this is far from properly describing the behaviors observed in animals and humans, this argument rejects the possibility of AIF as a candidate theory of everything in psychology. Several counterarguments have been advanced in response. First, it is argued that the prior preferences distribution $p(o)$ of biological agents is naturally biased and cannot be overwritten through learning. These biological prior preferences could for instance encode the desire to eat, by assigning low probabilities to the interoceptive observation caused by hunger. Second, the expression of the EFE contains an epistemic value term that encourages exploration and would drive the agent to avoid staying in the dark room. Though, as we have said, this epistemic value term might not be in direct connection with the original VFE objective function, which would in turn make this argument inadmissible. Third, real-world environments are actually dynamic and there might not be any real dark room. Staying in a region with totally predictable observations might be impossible, due to dynamic interoceptive inputs (if I do not eat, I will feel hungry), or simply external dynamics caused by the environment and other agents. Interestingly, (Berseth et al., 2021) have shown that an RL agent with an intrinsic drive to minimize surprise, and no other form of reward, could learn complex behaviors in environments that are naturally unstable.

6.2.4 Direct minimization of prediction error

Other related approaches have suggested directly minimizing surprise or prediction error using BPTT, without relying on a free-energy-based objective function. Based on target sensory observations and a generative model trained to perform sensory prediction, it is possible to infer latent states of the generative model using prediction error regression (Ahmadi and Tani, 2017). Action can be added to this scheme in different ways, which are represented in figure 6.2. In (Ahmadi and Tani, 2017), the target observations are directly provided in the proprioceptive space, and thus actions simply correspond to proprioceptive predictions. This could be implemented using any RNN architecture predicting motor commands (or actions), as represented in figure 6.2a. In (Otte et al., 2017; Butz et al., 2019; Jung et al., 2019), action is considered as a latent variable of generative models predicting sensory (exteroceptive) observations, as represented in 6.2b. Using target sensory states and BPTT, it is possible to infer actions that minimize future prediction error. Finally, (Mochizuki et al., 2013; Hwang et al., 2020; Matsumoto and Tani, 2020; Ohata and Tani, 2020) consider generative models of the form represented in figure 6.2c. These models can be trained to jointly predict motor commands and associated sensory observations by interacting with the environment, either using enforced motor trajectories (kinesthetic teaching) or motor babbling. For planning, the model can backpropagate prediction error information coming from the sensory level to infer the hidden state of the generative model. Motor commands corresponding to the desired sensory outcomes are then predicted using this inferred hidden state. All three of these conceptual architectures can be used to find motor commands associated with desired sensory outcomes, but only the last two can support indirect supervision not relying on target proprioceptive states.

The drawback of these models is that actions are driven by target sensory observations that are not a by-product of the agent's biased generative model, as suggested by AIF. To better anchor these approaches in the AIF framework, we suggest in section 6.5 including a separate generative model for desired sensory states that can act as a rail guiding learning and inference of motor commands.

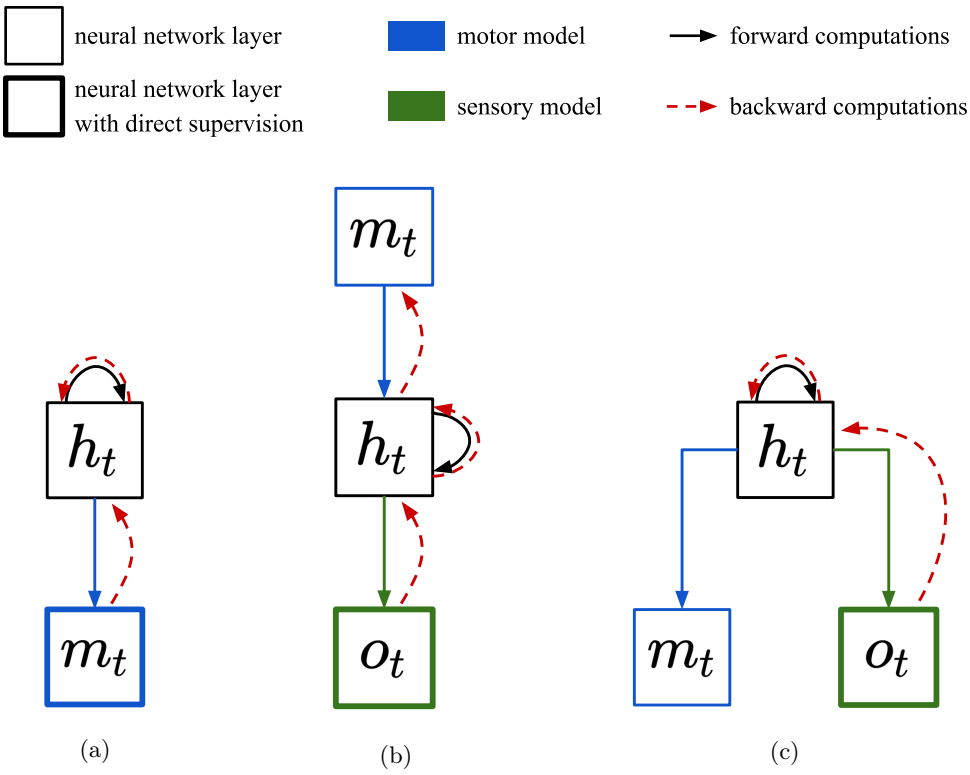


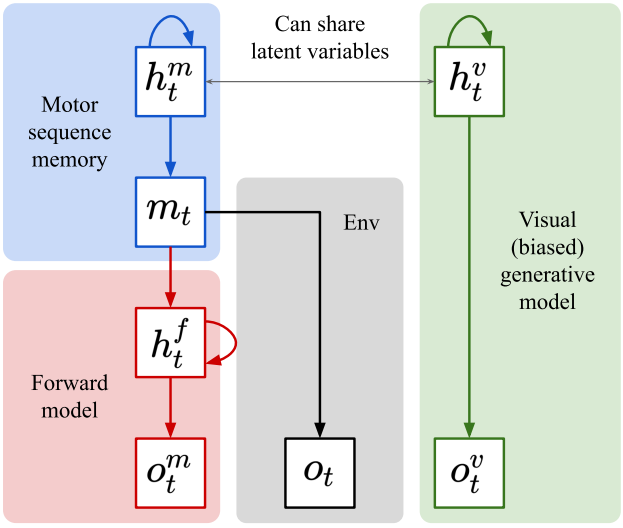
Figure 6.2: Different conceptual architectures using BPTT to find motor commands associated with minimal prediction error. Bold squares denote the modality (visual or motor) on which supervision is available in the form of target values. The red-dashed arrows represent the BP of prediction error for inference of the latent variables.

6.3 Proposed framework

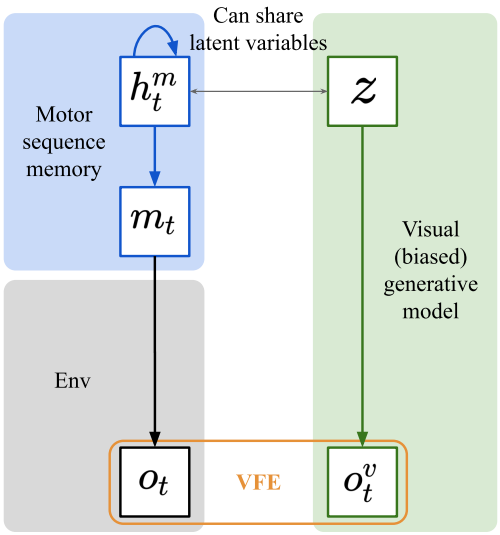
Our objective in this chapter is to extend AIF in order to learn a long-term memory of motor temporal patterns. So far, we have seen how AIF could cast surprise minimization as a drive for action, making it possible to retrieve motor commands corresponding to desired sensory observations predicted by a biased generative model. To build a sequence memory of motor patterns, our architecture needs to include a generative model of motor patterns. The models following the general structure described in figure 6.2c already include such a generative model. However, our intuition is that the forward model formulation of AIF (represented in figure 6.2b) is more natural and might be more powerful.

As such, we build upon this general structure and add another RNN generating sequences of motor commands $\boldsymbol{m}_t$. Additionally, we have expressed that to better align with the AIF framework, the agent should be equipped with a biased generative model for sensory observations. Adding these two components, we propose the general architecture represented in figure 6.3a.

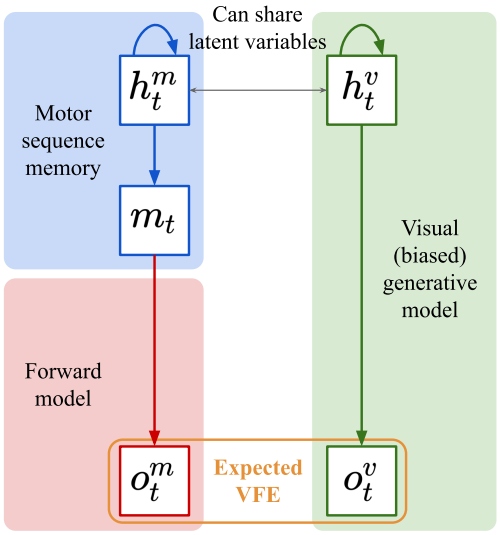
In the next sections, we present our attempts at implementing this framework for the learning of motor sequence memories, as well as for the dynamic control of motor trajectories. The two models we build can be seen as simplified versions of this general framework, as represented in figures 6.3b and 6.3c. The first attempt proposes a learning algorithm able to build a repertoire of motor trajectories using random exploration through goal-babbling. In the second attempt, the visual (biased) model is first trained to generate a repertoire of target trajectories, that we suppose are provided in the visual space by a teaching agent. Then, using only internal simulations (planning), the described architecture is able to learn a repertoire of corresponding motor trajectories. Additionally, this model is equipped with control mechanisms allowing to dynamically correct the motor trajectories in the presence of external perturbations. The next two sections provide the detailed methods and experiments for these two architectures.



(a) General framework.



(b) Proposed architecture for the learning of a motor sequence memory (Annabi et al., 2020).



(c) Proposed architecture for the learning and dynamic control of a motor sequence memory (Annabi et al., 2021b).

Figure 6.3: Proposed AIF architectures.

6.4 Autonomous learning of motor trajectories with AIF

The first work we present in this chapter does not implement planning for action selection. Instead, actions or action sequences, are corrected a posteriori after observations of their sensory consequences. This work thus puts a stronger focus on learning, and aims at training a long-term memory of motor sequential patterns, with no consideration about the adaptation properties that could be brought by dynamic inference of actions.

We consider an autonomous learning setting where no external supervision, in the form of reward or demonstration, is available to the agent. We explore the problem of building a repertoire of motor trajectories in a task-agnostic fashion, using an intrinsic drive for the agent to learn a repertoire that bests cover its state space. We suppose that a suitable repertoire of motor trajectories is one that can enable the agent to reach a diverse set of states.

To build this repertoire, the agent must at the same time learn a discrete set of states that properly cover its state space, and the motor trajectories to reach these states. We experiment with this idea using a simple handwriting environment, where a 2-DOF robotic arm draws simple trajectories. The agent controls the joint angle positions of the robotic arm, and perceives its environment through visual observations corresponding the the resulting drawing. We thus assume that the agent's environment is fully observable, and propose to perform direct perceptual inference using a Kohonen self-organizing map (Kohonen, 1982). We show how the representation provided by a Kohonen map can be seen as a categorical recognition probability distribution that minimizes the accuracy term in the expression of VFE. Additionally, the Kohonen map builds a repertoire of states that properly covers the state space of the agent, making this model particularly fit for our problematic of motor repertoire learning.

The agent learns motor trajectories reaching these states by optimizing the initial state of an RNN generating a sequence of motor commands, taking inspiration from the learning algorithm presented in the INFERNO model (Pitti et al., 2017). The choice of state to reach is random and renewed at each trial, the agent thus implements a form of goal babbling (Jacquey et al., 2019) to guide the exploration of its state space and the development of its motor skills. Contrary to models using the EFE as an objective function, we do not perform planning and only optimize the action plan a posteriori, through a learning method that minimizes the VFE objective function.

6.4.1 Methods

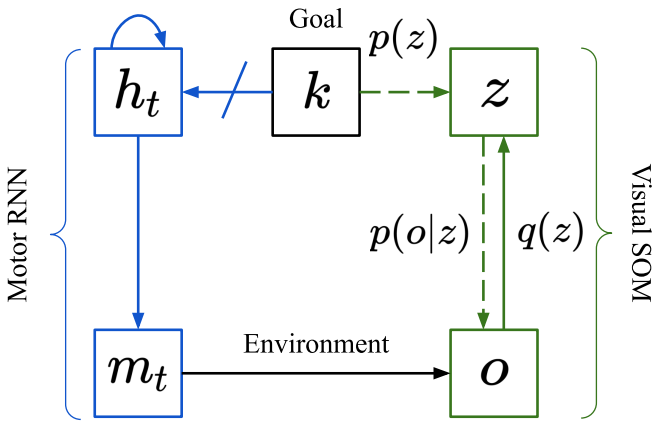


Figure 6.4: Architecture for the autonomous learning of motor trajectories.

Figure 6.4 represents the proposed architecture. The dashed lines represent implicit probability distributions between the variables that are not directly related to neural computations. The agent first samples an index k uniformly from its repertoire of motor trajectories. The RNN takes as input at time $t = 1$ the one-hot vector activated on the k-th dimension. In other words, the initial hidden state of the RNN $\boldsymbol{h}_1$ corresponds to the k-th column of the RNN input weights. The RNN generates and executes a sequence of motor commands $(\boldsymbol{m}_1, \cdots, \boldsymbol{m}_T)$. The resulting observation $\boldsymbol{o}$, at the end of the motor sequence, is provided as input to the Kohonen network, that finally

classifies this observation into a categorical representation z_i.

We consider two mechanisms that implement free-energy minimization: perceptual inference in the Kohonen network, and learning of the motor RNN input weights.

Kohonen network: Kohonen maps are a type of neural networks parameterized by a set of n input prototypes. When presented with an observation $\boldsymbol{o}$, such networks compare the distances of all the prototypes with regard to $\boldsymbol{o}$ and activate the neuron in the output layer $\boldsymbol{z}$ whose index is the index of the prototype closest to $\boldsymbol{o}$, denoted i^* . Learning in such networks consists in updating the set of prototypes into a direction that better describes the current observation $\boldsymbol{o}$. The Kohonen map used in our architecture is described by the following equations:

$$i^* = \underset{i<n}{\operatorname{argmin}}(\|\boldsymbol{W}_i - \boldsymbol{o}\|_2^2) \tag{6.3}$$

$$\boldsymbol{W} \leftarrow (1-\lambda)\cdot \boldsymbol{W} + \lambda \cdot \boldsymbol{N}(i^*) \odot \boldsymbol{o} \tag{6.4}$$

where $\boldsymbol{W}$ is a matrix of dimension (n, d_o) and n is the both the size of the repertoire of Kohonen prototypes and the size of the motor trajectories repertoire. The learning rule is weighted by a learning rate parameters λ. The neighborhood function $\boldsymbol{N} : [0, n-1] \rightarrow [0,1]^n$ also ponders this update rule for each index i, depending on $i*$. This function can be used to endow the Kohonen map with a certain topology. In our case, we consider a 1-d cyclic topology. The neighborhood function $\boldsymbol{N}(i^*)$ is maximum on the dimension $i = i^*$ and decreases exponentially according to the distance with regard to the winner neuron index i^*. This exponential decay is parameterised by a neighborhood width σ_k^2.

Free-energy derivations:

We use free-energy minimization as the strategy to train the input weights. What follows is a formalization of our model using a variational approach:

- $p(z)$ is the prior probability over states. Here we propose using a softmax, parameterised by $\beta > 0$, around the sampled index k.

$$p(Z = z_i) = \frac{\exp(-\beta|k-i|)}{\sum_j \exp(-\beta|k-j|)} \tag{6.5}$$

- $p(\boldsymbol{o}|z)$ is the state observation mapping. We suppose that the observation is an image composed of d_o pixels. For simplicity, we make the approximation of considering all pixel values as independent. We choose to use Bernoulli distributions for all pixel values $o_{l<d_o} \in \{0,1\}$. Since all pixel values are considered independent, the probability distribution over the whole observation can be factored as:

$$p(\boldsymbol{o}|Z = z_i) = \prod_{l<d_o} W_{i,l}^{o_l}(1-W_{i,l})^{1-o_l} \tag{6.6}$$

  where $W_{i,l}$ is the value of pixel l of the filter i of the Kohonen map.

- $q(z)$ is the recognition probability distribution over states, and depends on the observation $\boldsymbol{o}$. Here we define $q(z)$ to be the one-hot distribution over states such that $q(Z = z_i) = \delta_{i^*,i}$, where i^* is the index of the Kohonen neuron with the highest activation (i.e. whose filter is the closest to the observation).

We can now derive the free-energy computations using this model:

$$F(\boldsymbol{o}) = D_{KL}(q(z)||p(z)) - \sum_{i<n} q(z_i)\log(p(\boldsymbol{o}|z_i)) \tag{6.7}$$

$$= \sum_{i<n} q(z_i)\log\frac{q(z_i)}{p(z_i)} - \sum_{i<n} q(z_i)\log(p(\boldsymbol{o}|z_i)) \tag{6.8}$$

$$= -\log(p(z_{i^*})) - \log(p(\boldsymbol{o}|z_{i^*})) \tag{6.9}$$

$$= \beta|k-i^*| - \sum_{l<d_o}\left\{o_l\log(W_{i^*,l}) + (1-o_l)\log(1-W_{i^*,l})\right\} + C \tag{6.10}$$

where C is constant independent from the observation $\boldsymbol{o}$ and the recognition distribution category i^*.

The first term of the VFE in equation 6.7 is a quantity called complexity. It scores how complex the approximate posterior is compared to the prior. It decreases when $q(z)$ and $p(z)$ are close. In our case, it is minimal when $i^* = k$, meaning that the category chosen by the Kohonen map is the one with the highest prior probability. Minimizing complexity thus induces the motor RNN to generate trajectories that activate the right Kohonen category.

The second term is the (negative) accuracy. Accuracy measures how good the approximate posterior probability $q(z)$ is at predicting the observation $\boldsymbol{o}$. Here, it increases when the Kohonen filter of the winner neuron i^* is close to the observation. Maximizing accuracy induces the network to generate trajectories that are as close as possible to one of the Kohonen filter.

Summing those two quantities, minimizing VFE would result in observations that are close to one of the Kohonen filter, and in this Kohonen filter being the one with the highest prior probability. Using a variational formulation here allowed us to indirectly minimize surp without having to sum over all possible states $i < n$, although this could be tractable for small repertoires.

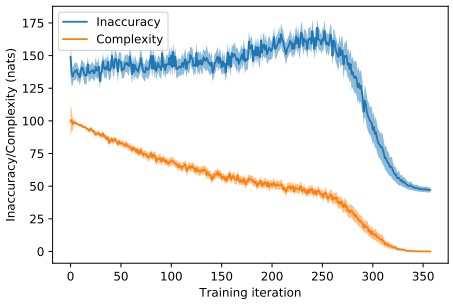
We can note that the Kohonen map performs a one-shot perceptual inference of the observation. Inferring the latent state corresponding to the filter closest to the input directly optimizes the accuracy term of the VFE, although it does not take into consideration the complexity term. As such, this perceptual inference process does not directly optimize VFE.

At each iteration, the agent samples an index k and performs the k-th motor trajectory of its repertoire. The input weights $\boldsymbol{W}_i$ are optimized following a random search attempting to minimize the VFE expression we have derived.

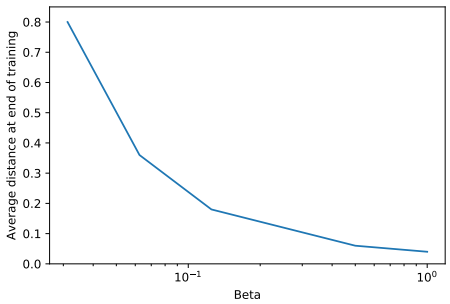
6.4.2 Results

The source code containing the implementation and training of this model is available on GitHub[1].

We trained our model for $E = 20000$ iterations on $n = 50$ primitives. At each iteration, we uniformly sample k from $[0, n-1]$. We train on the k-th primitive by adjusting the prior probability as in (6) and optimizing $\boldsymbol{x}_k$. On average, each activation signal $\boldsymbol{x}_k$ is trained on 400 iterations. During this training, we gradually decrease the Kohonen width σ_k and the random search variance σ^2.



(a) Inaccuracy and complexity averaged on the number of primitives $n = 50$, with $\beta = 8$. Each primitive has been sampled on at least 360 iterations, and on average on 400 iterations.



(b) Average distance $|i^* - k|$ between the activated neuron in the Kohonen i_w and the primitive index k, according to β.

Figure 6.5a displays the evolution of inaccuracy (negative accuracy) and complexity during training.

During the first phase, when $e < 12500$, the random search has a very high variance. Consequently, the trajectories generated and fed to the Kohonen map are very diverse and this allows the Kohonen map to self-organize. Inaccuracy does not seem to decrease in this early phase. This is because the Kohonen filters, initially very broad, are becoming more precise. The high variance in the random search allows a decrease of complexity but still generates trajectories that are too noisy to accurately fit the more precise Kohonen filters.

[1] https://github.com/sino7/motor_sequence_learning_som_aif

During the second phase, we decrease the variance of the random search. The system can now converge more precisely and this causes a faster decrease of both inaccuracy and complexity.

We notice that the inaccuracy cannot decrease below a certain value. At first, we could think that this is because the optimization strategy is stuck in a local optimum. However, we obtained the same lower bound on inaccuracy over different training sessions. Since the optimization strategy relies on random sampling, there is no evident reason to encounter the same local minimum. Our explanation is that this lower bound is imposed by the Kohonen network neighborhood function. Because the Kohonen width does not reach 0, the Kohonen prototypes are still attracting each other and this prevents them from completely fitting the presented observations. In consequence, the filters are always partly mixed with their neighbors and this causes the inaccuracy to plateau at a value that depends on σ_k^2.

Figure 6.5b shows the impact of the parameter β over the convergence. Looking at the derived expression for the VFE, we can see that this parameter directly scales the overall complexity. For low values of β, the random search is more likely to be stuck in local minima of free-energy, when activating a Kohonen neuron closer to k corresponds to an increase in inaccuracy that exceeds the decrease in complexity. We measured the average distance between the activated neuron in the Kohonen and the primitive index k at the end of training for $\beta \in \{2^{-5}, 2^{-4}, 2^{-3}, 2^{-2}, 2^{-1}, 2^{0}\}$. The results, presented in figure 6.5b, confirm that the final states obtained with higher values of β correspond to a more precise mapping between i^* (winner index of the Kohonen map) and k (state index enforced by the prior probability).

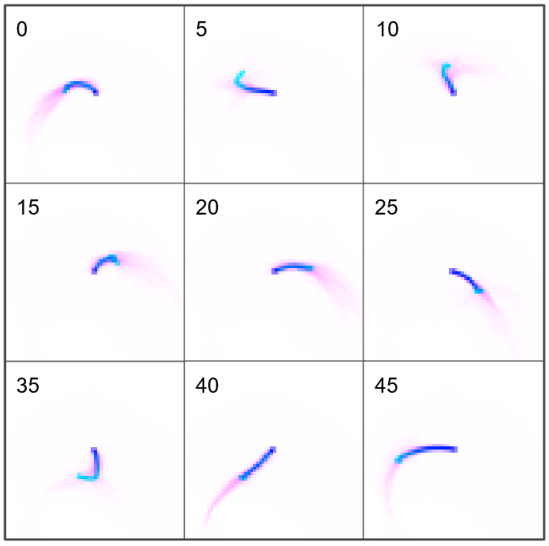


Figure 6.6: 9 of the 50 learned motor primitives and corresponding Kohonen filters: $k \in \{0, 5, 10, 15, 20, 25, 35, 40, 45\}$. The blue component of the image corresponds to the trajectory that is actually being generated by the network for the activation signal $\boldsymbol{x}_k$. The red component of the image corresponds to the Kohonen filter of index k.

Figure 6.6 displays some of the learned motor primitives. The blue component of the image corresponds to the trajectory that is actually being generated by the reservoir network for the activation signal $\boldsymbol{x}_k$. The red component of the image corresponds to the Kohonen filter of index k. This figures allows visual confirmation of several points. First, the inaccuracy at the end of training seems to indeed come from the blurriness of the Kohonen filters. Second, the filters and motor primitive trajectories seem to follow a topology: the index of the primitive seems highly correlated to the orientation of the route taken by the arm end effector. Finally, every trajectory seems to be in the center of the corresponding Kohonen filter, which suggests that the minimization of complexity successfully enforced the mapping between i^* and k.

6.4.3 Discussion

This learning method is interesting from a developmental point of view since it implements goal-babbling. Developmental psychology tells us that learning sensorimotor contingencies (O'Regan

and Noë, 2001) plays a key role in the development of young infants. In (Jacquey et al., 2019), the authors present a review about sensorimotor contingencies in the fields of developmental psychology and developmental robotics, in which they propose a very general model on how a learning agent should organize its exploration of the environment to develop its sensorimotor skills. They suggest that the agent should continuously sample goals from its state space and practice achieving these goals. The work we present in this section aligns nicely with their suggestion, as our agent randomly samples goals from a discrete state space, and optimizes the motor sequences leading to these discrete states.

This first method addressing the question of motor trajectories learning does not implement variational inference using the PC method. It was formulated in the beginning of this thesis in an effort to conciliate previously studied learning algorithms (Pitti et al., 2017) with AIF. The major drawback of this method is that the architecture does not infer the motor commands, only learning is performed on the motor generative model. As we will see in the second proposed method, online inference of the motor commands can translate into motor control and dynamic adaptation properties for the motor model.

6.5 Dynamic control using visual supervision

Indeed, it would be interesting to build a long-term memory of motor commands that additionally provides control mechanisms allowing the motor trajectories to be adapted dynamically. The previously presented model did not propose any control mechanism of this sort, because the objective function was only defined at the end of the trajectory. Indeed, the VFE was only evaluated at the end of each trial, using the observation caused by the complete motor trajectory.

Another drawback of the previous method is the number of training iterations that are needed during training. Each training iteration requires interaction with the environment, which can be limited in realistic settings. Consequently, we would like to endow our architecture with the ability to perform internal simulations using a forward model. Such an architecture would be capable of planning a course of action considering its future consequences, using only internal models. This opens the possibility of constructing the motor temporal patterns to be stored in the sequence memory, using minimal interaction with the environment.

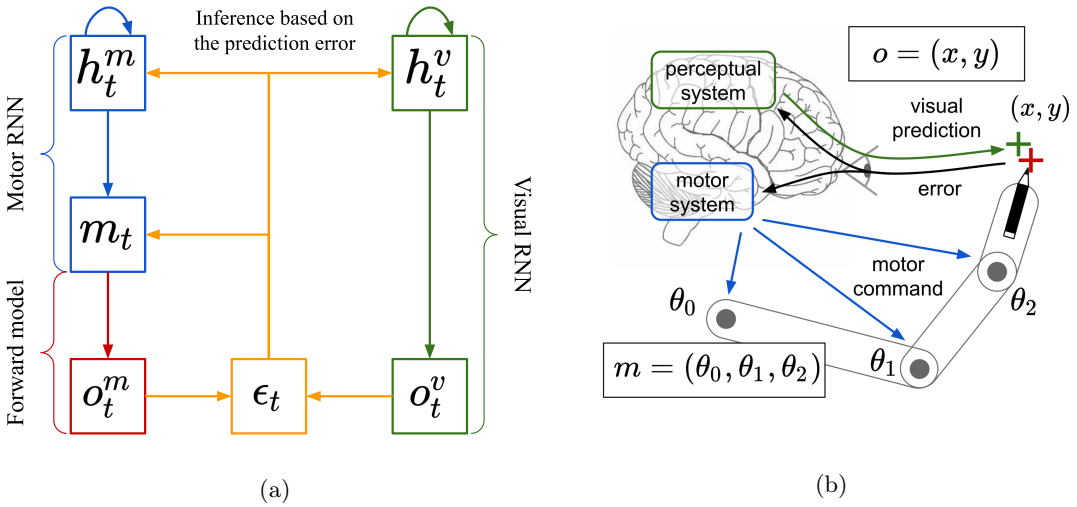
Contrary to the previous work, we assume here that the agent receives supervision from a teaching agent in the form of target trajectories in the sensory space $(\boldsymbol{o}_1^*, \cdots, \boldsymbol{o}_T^*)$. Building a repertoire of motor sequences is thus framed as an imitation learning problem, decomposed into several learning tasks. First, we have seen in the previous chapters how to learn a sequence memory for temporal patterns with direct supervision. This can be used to build a long-term memory of the visual trajectories provided by the teaching agent. Then, using internal simulations with a learned forward model, we show that we can build a sequence memory of motor patterns reproducing the target visual trajectories. Interestingly, we report that our architecture can model a bidirectional influence between the visual and motor models: motor commands can be dynamically inferred from the visual prediction, and reversely, visual prediction can be dynamically adapted from the predicted motor commands.

6.5.1 Methods

6.5.1.1 Architecture

Casting these ideas into the AIF framework, we propose to have two separate generative models predicting sensory observations, as represented in figure 6.7. Our embodied agent perceives information from its environment via visual observations that we denote $\boldsymbol{o}_t$, and can influence the state of the environment $\boldsymbol{h}_t$ via motor commands, denoted $\boldsymbol{m}_t$. We separate motor and visual pathways into two distinct systems interacting with each other only via a control mechanism minimizing prediction error on the visual level. Figure 6.7 displays an overview of our computational model for motor sequence learning.

In early stage of its development, we assume that our agent acquires a suitable forward model of its environment, denoted $\boldsymbol{f}$, predicting its visual observation based on its motor command and the previous state of the environment: $\boldsymbol{o}_t \sim \mathcal{N}\big(f(\boldsymbol{h}_{t-1}, \boldsymbol{m}_t), \sigma_o^2\big)$. Since our work does not focus on the learning of such a model, we omitted the dependency according to $\boldsymbol{h}_{t-1}$ to simplify the graph in figure 6.7a (for comparison, this dependency is represented in the general framework proposed



(a) (b)

Figure 6.7: Our bidirectional architecture for motor sequence learning and control.

in figure 6.3a).

Our agent's training is composed of the following stages :

- Learning of the visual generative model, predicting trajectories in the visual space and encoding the agent's preferences.
- Learning of a motor generative model according to the visual generative model and forward model.

The complete architecture features two parallel instances of the PC-RNN-HC-M model that we have derived in chapter 3. We have seen that this model can be trained to generate target temporal patterns. For the visual instance of this model, training can thus be performed using target trajectories provided in the visual space of the agent, for instance by a teaching agent. However, there is no direct supervision in the motor space for the learning of the motor model.

6.5.1.2 AIF using a forward model

The first generative model $p_v(\boldsymbol{o}_t)$ encodes the agent's biased beliefs about the environment. In our case, this generative model is trained in a preliminary step to reproduce a set of visual trajectories, supposedly provided by a teaching agent. The second generative model $p_m(\boldsymbol{o}_t)$ on the other hand, predicts sensory observations by the intermediary of motor commands $\boldsymbol{m}_t$:

$$p_m(\boldsymbol{o}_t, \boldsymbol{m}_t, \boldsymbol{h}_t^m) = p_m(\boldsymbol{o}_t|\boldsymbol{m}_t)p_m(\boldsymbol{m}_t|\boldsymbol{h}_t^m)p_m(\boldsymbol{h}_t^m) \tag{6.11}$$

Following the PC theory, our PC-RNN-HC-M instances encode at each layer the means of each recognition density. In other words, the activation of each layer represents the current best estimate of the latent variable using the recognition density. As such, the outputs of both the motor and visual pathways correspond to recognition density distributions $q_v(\boldsymbol{o}_t)$ and $q_m(\boldsymbol{o}_t)$. Thanks to the visual prediction, we can define the expected VFE for the motor model as:

$$EVFE_m(t) = \mathbb{E}_{q_v(\boldsymbol{o}_t)}\Big[\mathbb{E}_{q_m(\boldsymbol{m}_t,\boldsymbol{h}_t^m)}\big[-\log p_m(\boldsymbol{o}_t|\boldsymbol{m}_t, \boldsymbol{h}_t^m)\big] + D_{KL}\big(q_m(\boldsymbol{m}_t, \boldsymbol{h}_t^m|\boldsymbol{o}_t)||p_m(\boldsymbol{m}_t, \boldsymbol{h}_t^m)\big)\Big] \tag{6.12}$$

Reciprocally, the motor model also predicts future sensory observations, which makes it possible to define the expected VFE for the visual model as:

$$EVFE_v(t) = \mathbb{E}_{q_m(\boldsymbol{o}_t)}\Big[\mathbb{E}_{q_v(\boldsymbol{h}_t^v)}\big[-\log p_v(\boldsymbol{o}_t|\boldsymbol{h}_t^v)\big] + D_{KL}\big(q_v(\boldsymbol{h}_t^v|\boldsymbol{o}_t)||p_v(\boldsymbol{h}_t^v)\big)\Big] \tag{6.13}$$

For simplicity, we assume that $q_v(\boldsymbol{o}_t)$ and $q_m(\boldsymbol{o}_t)$ are Dirac delta distributions characterized by the parameters $\boldsymbol{o}_t^v$ and $\boldsymbol{o}_t^m$, corresponding respectively to the observations predicted at time t

by the visual system and motor system. This greatly simplifies the computations of the outside expectations in equations 6.12 and 6.13. We obtain the following expressions:

$$EVFE_m(t) = \mathbb{E}_{q_m(\boldsymbol{m}_t, \boldsymbol{h}_t^m)}\big[-\log p_m(\boldsymbol{o}_t^v | \boldsymbol{m}_t, \boldsymbol{h}_t^m)\big] + D_{KL}\big(q_m(\boldsymbol{m}_t, \boldsymbol{h}_t^m | \boldsymbol{o}_t^v) || p_m(\boldsymbol{m}_t, \boldsymbol{h}_t^m)\big) \quad (6.14)$$

$$EVFE_v(t) = \mathbb{E}_{q_v(\boldsymbol{h}_t^v)}\big[-\log p_v(\boldsymbol{o}_t^m | \boldsymbol{h}_t^v)\big] + D_{KL}\big(q_v(\boldsymbol{h}_t^v | \boldsymbol{o}_t^m) || p_v(\boldsymbol{h}_t^v)\big) \quad (6.15)$$

These equations directly correspond to the expression of the VFE when the observation $\boldsymbol{o}_t^*$ is provided (observed). Basically, the proposed AIF implementation suggests using the observation predicted from the visual pathway as target for the motor pathway, and vice versa.

Since the visual pathway is only composed of a PC-RNN-HC-M model, minimization of EVFE_v is naturally ensured by the dynamics of the network, provided that we use the observations predicted by the motor pathway as targets.

For the motor pathway, it is slightly more complex since prediction error information has to flow bottom-up through the forward model. Since in this work we do not focus on the learning of the forward model, we also disregard the associated local PC update rules. Instead, we directly use the (supposedly known) gradient of the forward model to correct the motor command based on the prediction error. The corrected motor command $\boldsymbol{m}_t^*$ is thus computed as a gradient descent update on $\boldsymbol{m}_t$:

$$\boldsymbol{m}_t^* = \boldsymbol{m}_t - \alpha_m \nabla_{\boldsymbol{m}_t} \|\boldsymbol{o}_t^m - \boldsymbol{o}_t^v\|_2^2 \quad (6.16)$$

$$= \boldsymbol{m}_t - \alpha_m \nabla_{\boldsymbol{m}_t} \|\boldsymbol{f}(\boldsymbol{m}_t) - \boldsymbol{o}_t^v\|_2^2 \quad (6.17)$$

In opposition to optimal control theory, the corrected motor command $\boldsymbol{m}_t^*$ is not obtained through the use of an inverse model taking as input the target observation $\boldsymbol{o}_t^v$. Instead, this value is inferred through a one-step gradient descent update using the forward model.

It is also worth noting than in analogy with the different implementations of feedback connections in the PC literature, we could also consider here random or learned feedback synaptic weights. Additionally, the forward model can very well comprise several layers, or other input variables (such as a representation of the state of the environment). The PC scheme can be applied to arbitrarily deep networks and the prediction error information transported bottom-up into the hierarchy.

As a proof of concept, we assume here that the forward model learned by the agent perfectly simulates the environment, and that the associated feedback connections perfectly compute the gradient of the forward model.

6.5.2 Results

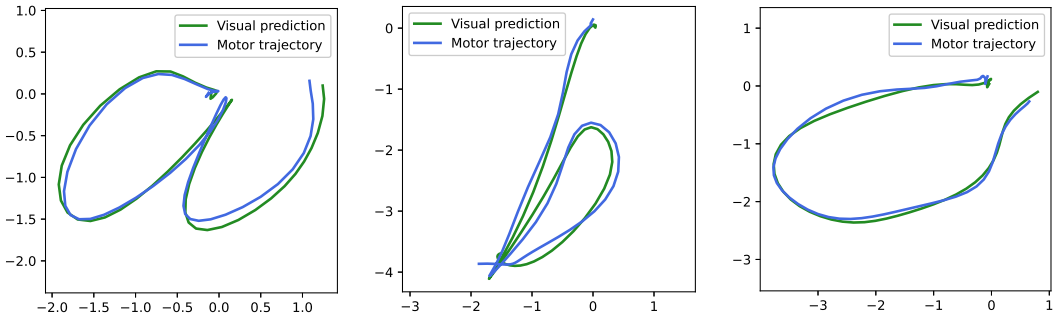
In this subsection, we experiment with the complete architecture presented in figure 6.7. Figure 6.7b represents our experimental set up for this experiment. The agent evolves in an environment with which it can interact through sensors and actuators. Since we focus here on motor skill learning, we simplified the visual space of our agent by already decoding the position of the agent's end-effector from its visual input. In other words, the agent directly receives as visual input the position of its end-effector in Cartesian coordinates. The agent acts on the 2D environment by moving the 3-DOF simulated arm. The extremity of the first joint (the shoulder) is fixed on the position (-6, 6). The three joints are of respective lengths 6, 4 and 2. We do not cover here the learning of the forward model, and suppose that the agent has learned through motor babbling how its actions influence its observations. In our experiments, the forward model is replaced by the real physical model outputting end-effector positions in Cartesian coordinates according to joint orientations. Our architecture's training is thus composed of two stages:

1. Supervised learning of visual handwritten trajectories : The RNN predicting trajectories in the visual space (2D) is trained using the online learning rules described in chapter 3.

2. Training of the motor RNN : The RNN generating trajectories in the motor space (3 dimensions) is trained using the method described in section 6.5.1.2, using the trajectories predicted by the visual RNN as indirect supervision to perform AIF.

The parameters used to obtain the results presented in this section are provided in appendix E. The source code is available on GitHub[2].

6.5.2.1 Motor PC-RNN-HC-M learning



(a) (b) (c)

Figure 6.8: Generated motor trajectories and predicted visual sequences of 2D positions at the end of training for the three given classes a (left), b (middle) and c (right). Results obtained with $n = 50$ and $p = 3$.

The visual prediction RNN generates a trajectory of observations that the motor RNN tries to replicate. We train the motor RNN according to the method detailed in section 6.5.1.2. The motor RNN hidden states are initialized randomly and shared across the different classes of trajectories. The initial hidden causes are one-hot vectors of dimension p (where p is the number of motor trajectories to write in the memory) encoding the trajectory label. Figure 6.8 displays learned motor trajectories along with the corresponding predicted visual trajectories. These results were obtained with a hidden state dimension of $n = 50$ for both RNNs and with $p = 3$ classes, after 40000 iterations. Training is arguably long with regard to the difficulty of the proposed task. We suppose that this is due to the PC learning mechanism, that can only approximate BP properly if we let enough time for the inference of each variable in the computational graph to converge.

6.5.2.2 Model capacity comparative analysis

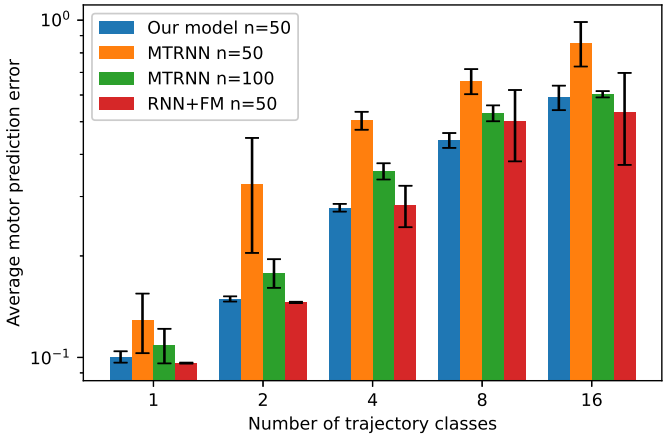


Figure 6.9: Comparison of the reconstruction error according to the number of trajectory classes for four models : our model with $n = 50$, two instances of the MTRNN model proposed in (Mochizuki et al., 2013) with $n = 50$ and $n = 100$, and the RNN+FM model with $n = 50$. The columns display the average motor prediction error on the testing data set. Intervals in black indicate confidence intervals.

[2]https://github.com/sino7/bidirectional-interaction-between-visual-and-motor-generative-models

Before analyzing other behaviors of the proposed model, we validate it by comparing its performance with two benchmark models.

First, we compare our performance with the method presented in (Mochizuki et al., 2013), proposed on a similar task. This method uses a Multiple Timescales RNN (MTRNN) (Yamashita and Tani, 2008) model for the joint generation of visual and motor trajectories as represented in figure 6.2c. The model is first trained using visuomotor trajectories obtained through motor babbling. Then, optimization of the initial MTRNN hidden state is performed for each target visual trajectory using BPTT. Finally, the MTRNN weights are tuned to ensure coherence between the visual and motor outputs.

Second, we compare our model with a simple architecture composed of an RNN generating motor trajectories and a forward model (equivalent to the one present in our model) translating this motor output into visual trajectories, as represented in figure 6.2b. Indeed, we could argue that using the forward model we introduced, one could simply backpropagate gradients originating from an error signal in the visual space to learn motor trajectories. We train such a model, that we label RNN+FM, using BPTT, and compare its performance with our model.

Performances of the three models are evaluated according to their precision on the motor trajectory, and their capacity to encode a large number of trajectories. Precision is measured with the average error on the testing data set, that is, the error between the position of the tip of the pen when performing the motor trajectory, and the target trajectory. Note that since the forward model $\boldsymbol{f}$ is perfect, the actual position when performing the motor trajectory is equal to the predicted outcome $\boldsymbol{o}^m = \boldsymbol{f}(\boldsymbol{m})$. We call this quantity *motor prediction error*, and it can be expressed as:

$$E_m = \frac{1}{T} \sum_{t \leq T} \| \boldsymbol{f}(\boldsymbol{m}_t) - \boldsymbol{o}^*_t \|_2^2$$

In the following sections, we will sometimes measure the error between the visual prediction and the target trajectory, we will call this quantity *visual prediction error*. It can be expressed as:

$$E_v = \frac{1}{T} \sum_{t \leq T} \| \boldsymbol{o}^v_t - \boldsymbol{o}^*_t \|_2^2$$

Figure 6.9 displays the evolution of the motor prediction error for our model, two instances of the MTRNN model, and the RNN+FM model.

If we extract the motor RNN model from our complete architecture, and compare it with an MTRNN model with the same hidden state dimension, both models have a comparable number of parameters. However, we could argue that the same MTRNN can generate trajectories in both visual and motor space, while our motor RNN only generates trajectories in the motor space. For fair comparison, we might want to include into the parameter count the parameters of the visual RNN in our architecture. For this reason, we extend the comparison with an MTRNN model with a state dimension of $n = 100$, with twice as many parameters as our two RNNs combined. Finally, note that our approach, contrary to the MTRNN model, assumes that a perfect forward model is available to perform AIF.

Still, the results displayed in figure 6.9 tend to show that our architecture can compete with other algorithms for motor trajectory learning with indirect supervision.

6.5.2.3 Intermittent control

One of our model's feature not discussed previously is the possibility to switch off the feedback pathway when prediction error is under a certain threshold. We experimented with this idea by varying such a threshold and observing when the feedback pathway would switch on and off. Figure 6.10 displays results that were obtained with a hidden state dimension of $n = 50$ with $p = 3$ classes, after training of the motor RNN.

For the lowest threshold value, the feedback pathway is almost always active and the motor trajectories are controlled to accurately match the predicted visual trajectories. For the highest threshold value, the feedback pathway never activates and the trajectory performed corresponds to the natural trajectory of the motor RNN. This mechanism is interesting as it could be used to control the trade-off between precision and smoothness of the generated trajectories in situations where the visual target trajectory is not smooth.

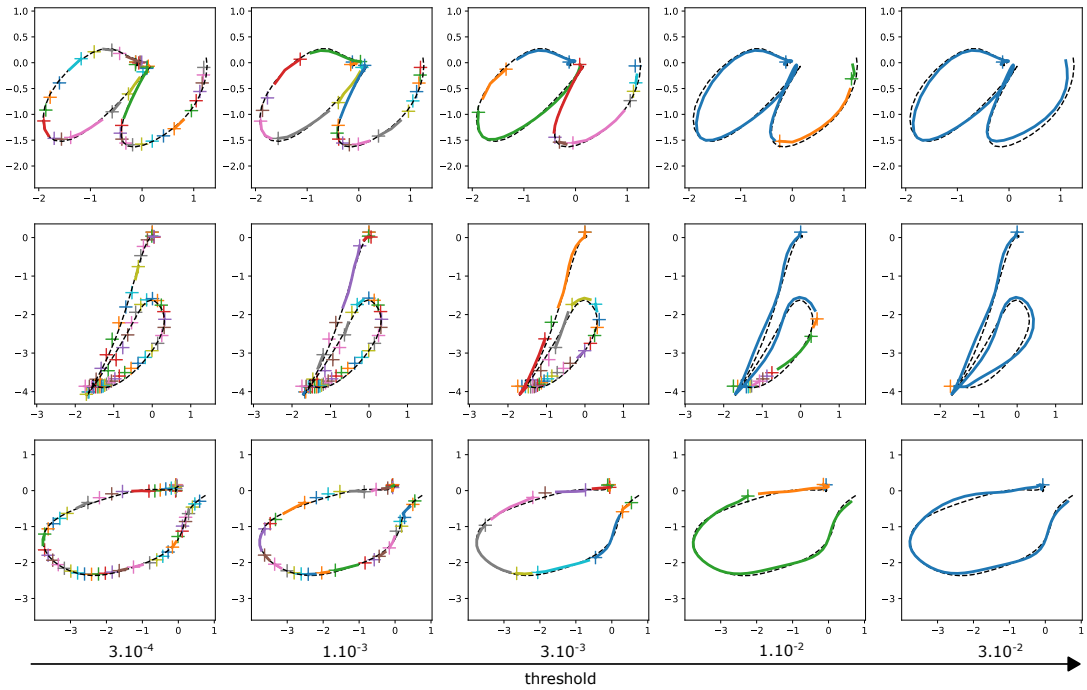
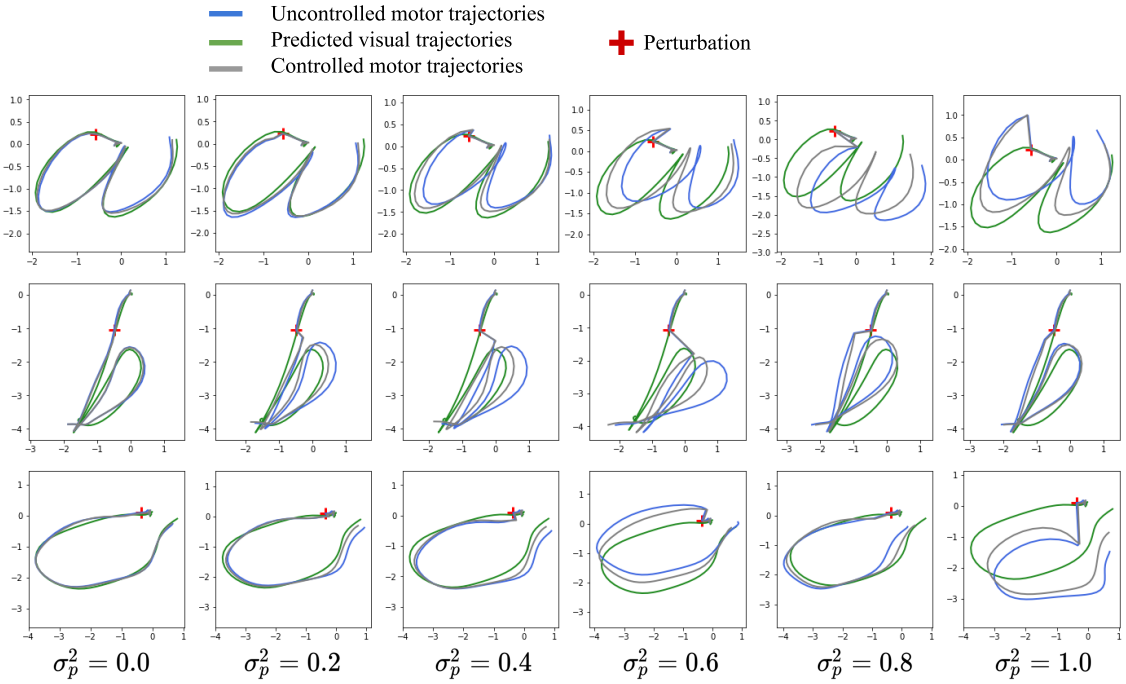
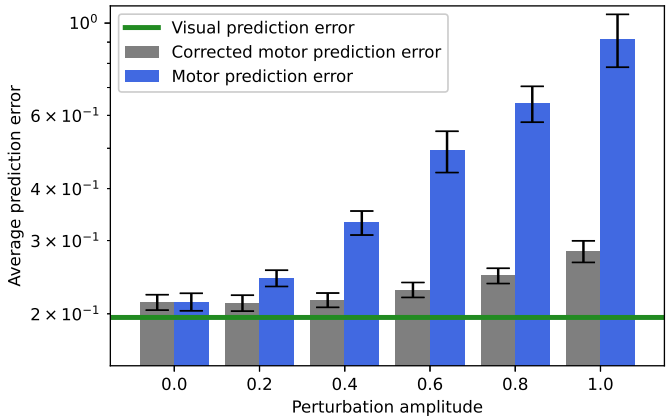


Figure 6.10: Trajectories generated by the motor RNN, with a state dimension of 50, displayed into the visual space. The black dashed line represents the visual trajectory predicted by the visual RNN. The trajectories generated by the motor RNN are represented as successions of trajectory segments, differentiated by their color. Each new trajectory segment corresponds to an activation of the AIF controller, and is represented by a plus shaped marker. The different figures correspond to different activation threshold values for the controller, from left to right: $3.10^{-4}, 1.10^{-3}, 3.10^{-3}, 1.10^{-2}, 3.10^{-2}$.

6.5.2.4 Robustness to external perturbations



(a) Examples of trajectories obtained in the perturbation experiment, for different perturbation amplitudes $\sigma_p^2 \in \{0.0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. Blue lines represent the uncontrolled motor trajectories displayed in the visual space. Green lines represent the predicted visual trajectories. Gray lines represent the corrected motor trajectories displayed in the visual space. These trajectories were obtained with an RNN hidden state dimension of 50 after training.



(b) Average prediction error according to the perturbation amplitude σ_p^2. Prediction errors are measured as average distances with regard to corresponding test trajectories of the same label.

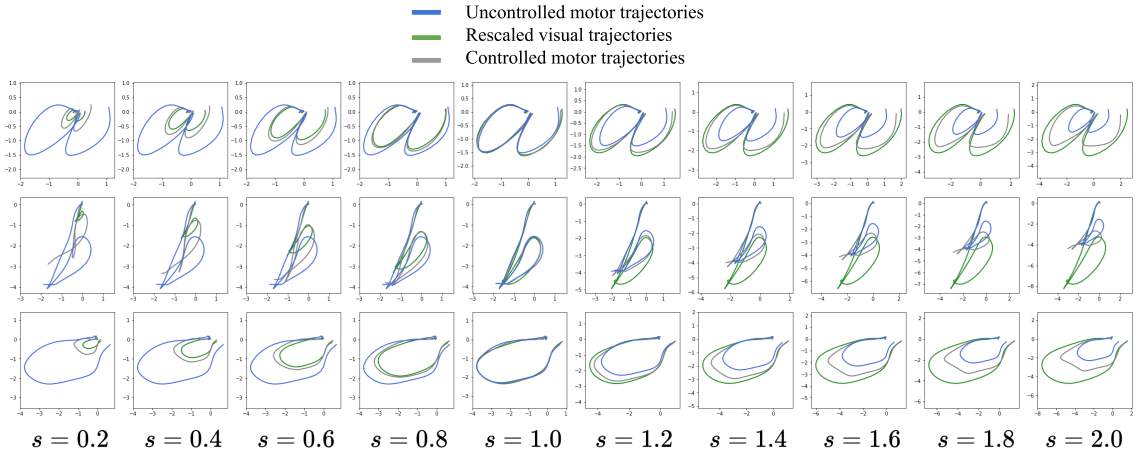
Figure 6.11: Perturbation robustness experiment. At the 10th time step, we apply a random perturbation, represented in red, on the motor output. Our model uses the visual prediction as a guide to correct the perturbed motor trajectory.

In this experiment, we consider applying external perturbations of variable amplitude onto the motor output of our model. For each trajectory, we sample a perturbation from a multivariate normal distribution of variance $\sigma_p^2 \mathbb{I}_3$. This perturbation is added to the motor output of the generative model for all timesteps $t > 10$. Figure 6.11 displays examples of obtained trajectories

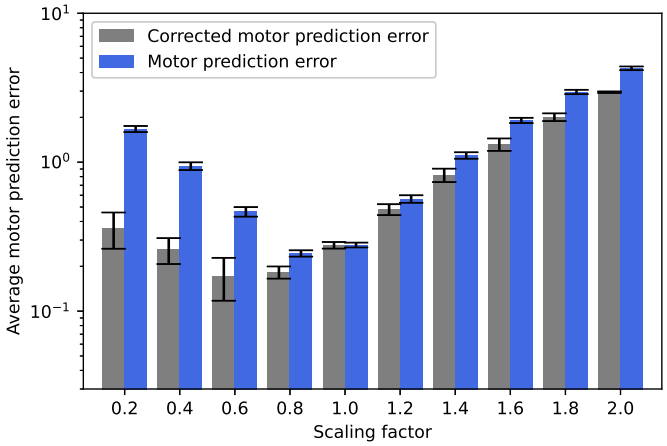
with or without control, and the evolution of the average motor prediction error with regard to the perturbation amplitude σ_p^2.

The visual prediction, represented in green, acts as a guide to dynamically control the motor trajectory, represented in blue. If we compare the average motor prediction error with and without control (respectively in grey and blue in figure 6.11b), we can observe that the control highly reduces the error brought by the perturbation. These results demonstrate that our model generates motor trajectories robust to external motor perturbations.

6.5.2.5 Adaptation to transformed visual predictions



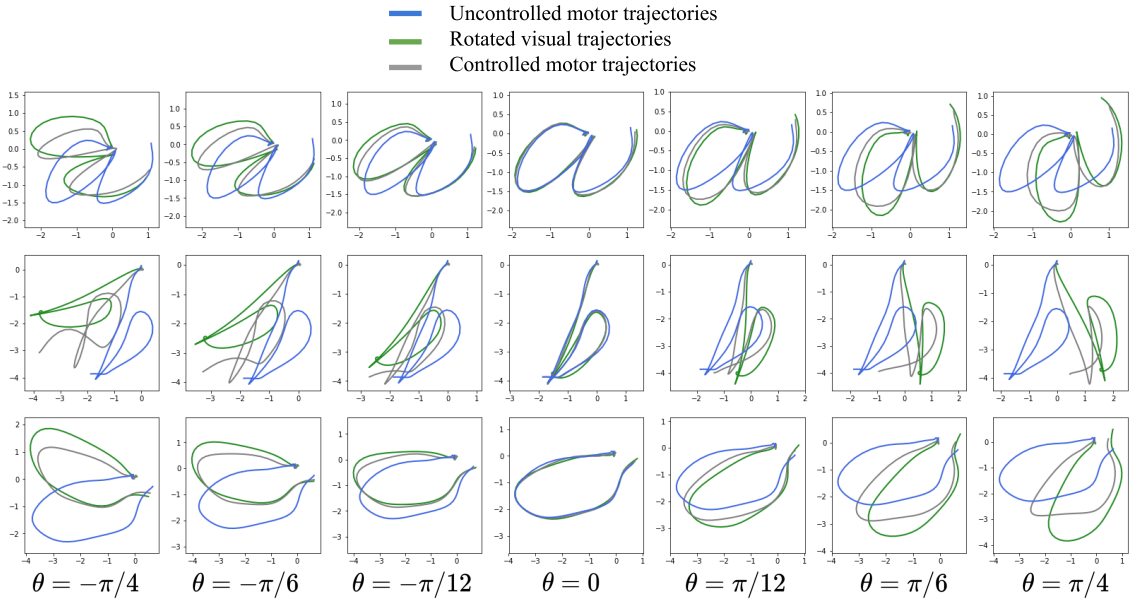
(a) Examples of trajectories obtained in the scaling experiment, for different scales $s \in \{0.2, 0.4, 0.6, 0.8, 1.0, 1.2, 1.4, 1.6, 1.8, 2.0\}$. Blue lines represent the uncontrolled motor trajectories displayed in the visual space. Green lines represent the rescaled visual trajectories. Gray lines represent the corrected motor trajectories displayed in the visual space. These trajectories were obtained with an RNN hidden state dimension of 50 after training.



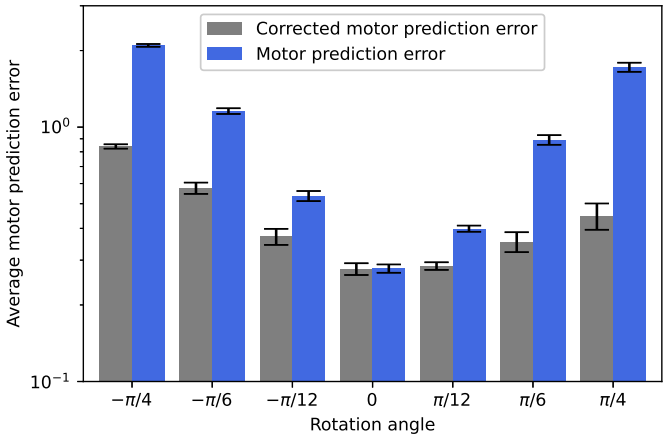
(b) Average prediction error according to the scaling factor applied to the visual prediction. Prediction errors are measured as average distances with regard to corresponding rescaled test trajectories of the same label.

Figure 6.12: Adaptation to scaling experiment. We apply a change of scale on the visual prediction. Our model dynamically adapts the motor trajectory to fulfill best the visual prediction.

In this experiment, we consider the situation where a transformation is applied to the predicted visual trajectory. Changes of scales and rotations are transformations that can be applied easily on the visual output. However, generating the motor commands performing these transformed visual



(a) Examples of trajectories obtained in the rotation experiment, for different rotation angles $\theta \in \{-\pi/4, -\pi/6, -\pi/12, 0, \pi/12, \pi/6, \pi/4\}$. Blue lines represent the uncontrolled motor trajectories displayed in the visual space. Green lines represent the rotated visual trajectories. Gray lines represent the corrected motor trajectories displayed in the visual space. These trajectories were obtained with an RNN hidden state dimension of 50 after training.



(b) Average prediction error according to the angle of the rotation applied to the visual prediction. Prediction errors are measured as average distances with regard to corresponding rotated test trajectories of the same label.

Figure 6.13: Adaptation to rotation experiment. We apply a rotation on the visual prediction. Our model dynamically adapts the motor trajectory to fulfill best the visual prediction.

trajectories is less trivial. We experiment here with applying changes of scales and orientations to the visual prediction while controlling the motor trajectory using the prediction error feedback. Results of those experiments for different scales and orientations are displayed respectively in figures 6.12 and 6.13.

Figure 6.12b displays the evolution of the error between the motor trajectories and the rescaled test trajectories, with and without control. We first notice that overall, reducing the scale seems to induce lesser prediction errors than increasing the scale. This is because the prediction error is computed as a path integral of point to point distances, thus sensible to the scale of the trajectories. Second, we notice that the improvement brought by the online control is less effective when increasing the scale. This is due to the fact that the motor RNN was trained to exhibit trajectories of limited amplitudes. Inference of the hidden state of the RNN is not sufficient to properly control the trajectory.

Figure 6.13b displays the evolution of the error between the motor trajectories and the rotated test trajectories, with and without control. We notice that the architecture manages to adapt better to rotations in the counterclockwise direction. This could be due to the fact that performing these trajectories requires smaller modifications on the natural motor trajectory because of the arm configuration.

6.5.2.6 Motor control comparative analysis

The previous experiments have shown that our model can perform online inference of hidden states in the motor RNN to dynamically adapt to motor perturbations and transformations of the visual target trajectory.

In this section, we provide a comparison with an other inference method, applied to our model and on the generative model described in figure 6.2c and previously used as baseline in section 6.5.2.2.

ERS (Ahmadi and Tani, 2017) has been proposed as a method to perform inference of hidden states in RNNs. Let us consider an RNN with hidden state $\boldsymbol{h}_t$ and output $\boldsymbol{o}_t$. According to the ERS, at each time step t the model predicts the next W time steps and computes the prediction error on the W predictions $(\boldsymbol{o}_t, \dots, \boldsymbol{o}_{t+W-1})$ with regard to targets $(\boldsymbol{o}^*_t, \dots, \boldsymbol{o}^*_{t+W-1})$. This error signal is then backpropagated to infer the value of $\boldsymbol{h}_t$. Finally, this regression of $\boldsymbol{h}_t$ can be iterated I times at each time step.

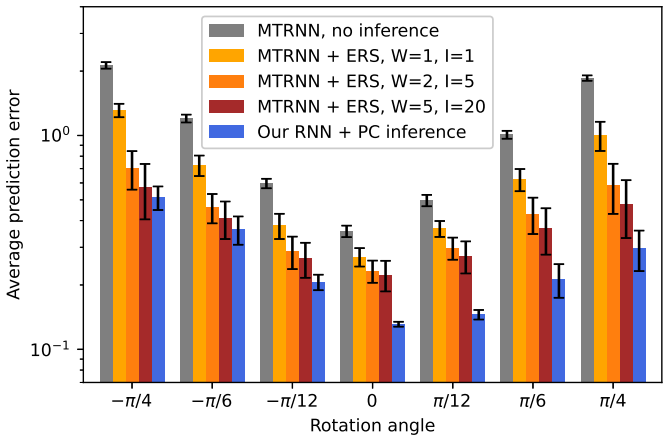
Comparing this inference method with the inference performed in our RNN implementing PC, the ERS will have a time complexity equivalent to $W \times I$ times the time complexity of our model, and a space complexity of W times the space complexity of our model. If $I = 1$ and $W = 1$, the ERS method is comparable with our inference method in terms of complexity. In fact, we can argue that our inference method closely relates to ERS as it also implements a gradient descent optimization. Still, they differ in several aspects. Our method minimizes VFE while ERS minimizes prediction error. Even though these two quantities are closely related, as shown in chapter 2, this still brings some differences in the two approaches. Additionally, our method does not rely on BPTT, and inference is part of the dynamics of the neural network.

In (Hwang et al., 2020), the authors use an RNN model for joint generation of visual observations and motor commands, and perform inference with the ERS based on visual targets. Iteratively using the ERS to infer the hidden state, and generating a motor command from this hidden state makes it possible to perform motor control on the full trajectory. We take inspiration from this work and implement ERS on the joint visuomotor MTRNN architecture introduced in section 6.5.2.2.

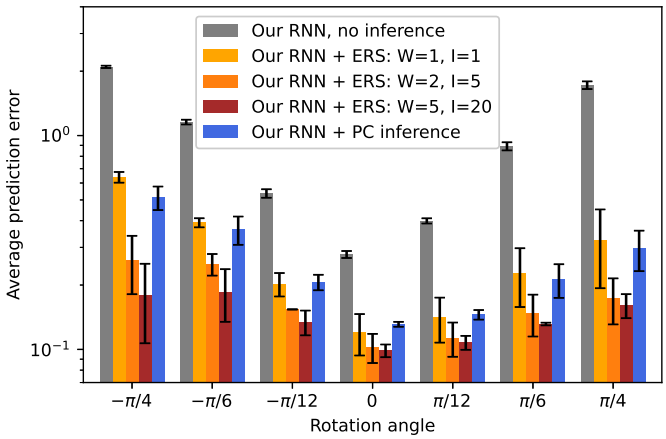
We consequently have two models that we can use for comparison: ERS applied to our motor RNN, and ERS applied to the joint visuomotor MTRNN.

We reuse one of the previous experimental set ups to perform this comparative analysis. The task is for both models to generate the motor trajectories corresponding to rotated visual targets $(\boldsymbol{o}^*_0, \dots, \boldsymbol{o}^*_{T-1})$. As in section 6.5.2.5, our model is trained to generate the corresponding motor trajectories with standard orientation ($angle = 0$). It has to dynamically adapt the motor trajectory with regard to the rotated visual target trajectory. We reuse the MTRNN implementation introduced in section 6.5.2.2. Similarly to our model, it is trained to generate the motor trajectories corresponding to the visual targets with standard orientation.

Figure 6.14 displays the motor prediction errors obtained with all three models, with different values for W and I for the ERS based control.



(a) Comparison between our inference method applied to our RNN (blue), and the ERS applied to the MTRNN model (grey, yellow, orange and red).



(b) Comparison between our inference method (blue) and the ERS (grey, yellow, orange and red), both applied to our RNN.

Figure 6.14: Comparison of the inference methods. The presented results aggregate the motor prediction errors over the p categories, $T = 60$ timesteps, for 5 instances of each model.

We first focus on the comparison between our approach and the ERS applied to the MTRNN model (figure 6.14a). According to these results, our approach outperforms the ERS applied to the MTRNN model, even when allowing a factor 10 increased time complexity ($W = 2$ and $I = 5$), and a factor 100 increased time complexity ($W = 5$ and $I = 20$). However, we cannot conclude whether this difference in performance is due to the RNN model or the inference method.

The second comparison can help answering this question by comparing the two inference methods on our RNN model. The results displayed in figure 6.14b show that our inference method performs very similarly to the ERS with parameters $W = 1$ and $I = 1$. This result is not very surprising, since we have explained that both approaches are closely related. However, when used with more iterations I and a longer time window W, the ERS outperforms our approach.

Overall, we can conclude that performing inference by propagating gradients backward through a forward model seems to be a better approach than performing inference on a single model for the joint generation of visual and motor trajectories. Still, this conclusion relies on the assumption that the forward model used in our experiments is perfect, the observed performance gap might not be as significant if we used a learned approximate forward model.

6.5.2.7 Reciprocal influence

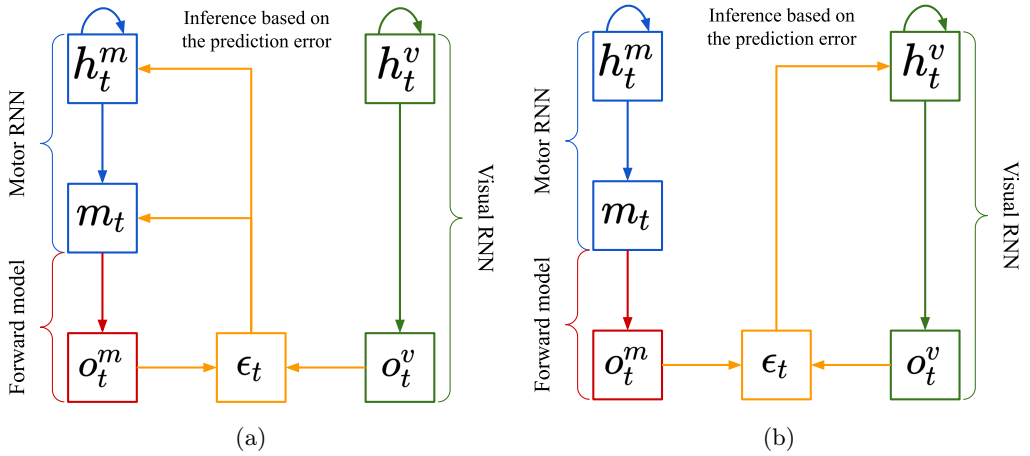


Figure 6.15: Illustration of the visual to motor feedback pathway (left) and the motor to visual feedback pathway (right).

In all the previous experiments of this section, we have considered only one feedback pathway between the two possible feedback loops in our architecture, represented in figure 6.15. Whether it was to learn motor trajectories, for intermittent control, to adapt to perturbations on the motor output, or to adapt to transformations on the visual prediction, we only used the visual to motor feedback pathway (figure 6.15a). Since the motor RNN was trained using the visual prediction RNN, using the motor predictions $\boldsymbol{o}^m$ to control for the visual predictions $\boldsymbol{o}^v$ would only result in less precise visual predictions.

However, the symmetry of the interaction between the two modalities can still be exploited in situations where the visual prediction RNN performs worse than the motor RNN. To create such situations, we impaired the visual prediction RNN by applying a multiplicative noise $\mathcal{N}(1, \sigma_i^2)$ to the model's parameters $\boldsymbol{W}_{out}$, $\boldsymbol{W}_p$, $\boldsymbol{W}_f$, $\boldsymbol{W}_c$. We argue that this situation can arise naturally if we consider the lifelong learning of an agent. The impairment we simulated here could correspond to the forgetting of the visual prediction RNN due to training on a different task.

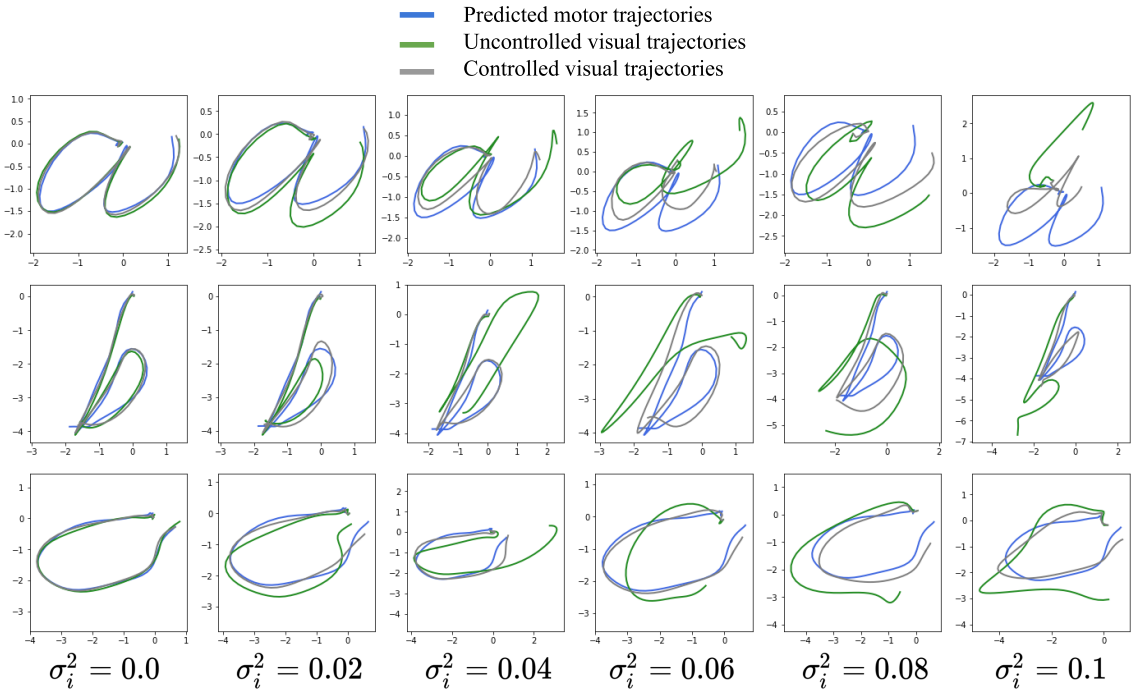
Similarly to the previous experiments, we display in figure 6.16 examples of predicted visual trajectories with or without correction from the motor RNN, and the evolution of the average visual prediction error with regard to the impairment amplitude σ_i^2.

We observe that visual predictions benefit from the correction brought by the motor predictions. In all the tested situations where the impairment amplitude is greater or equal than 0.02, the uncorrected visual prediction error (green) is higher that the corrected visual prediction error (grey). These results demonstrate that knowledge in our architecture can be transferred in both directions.

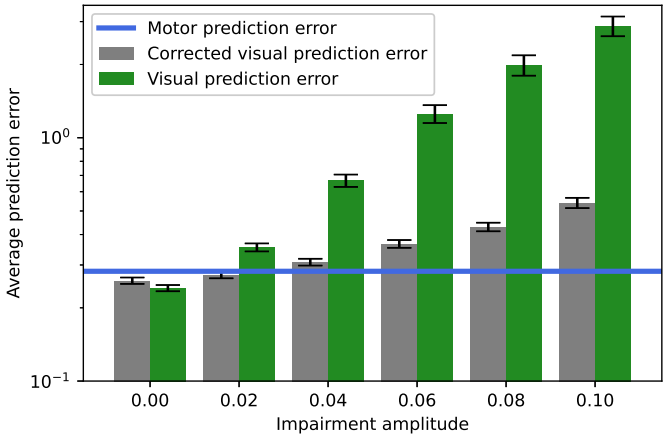
6.5.2.8 Bidirectional influence

We have so far considered situations where only one of the two feedback pathways is activated at a time. However, our model should in principle always allow propagation of the error in both directions. According to the FEP, the visual prediction $\boldsymbol{o}^v$, and the predicted visual outcome of actions $\boldsymbol{o}^m$, should both be associated with a standard deviation. If one prediction is more accurate than the other, the agent should put more trust in this prediction and direct the influence so that the less accurate model can adapt to the more accurate one. The interaction shall still remain bidirectional, but the influence in one direction will be stronger than in the other.

In practice in the previous experiments, we have assumed that the agent should trust one model more than the other, to the extent where we have turned off the reciprocal influence. During learning of the motor model, and for online adaptation to perturbations and changes of scales and orientations, the bidirectional interaction was approximated as a unidirectional influence from the visual model to the motor model. On the contrary, in the last experiment where we have intentionally impaired the visual model, the bidirectional interaction was approximated as a unidirectional influence from the motor model to the visual model.

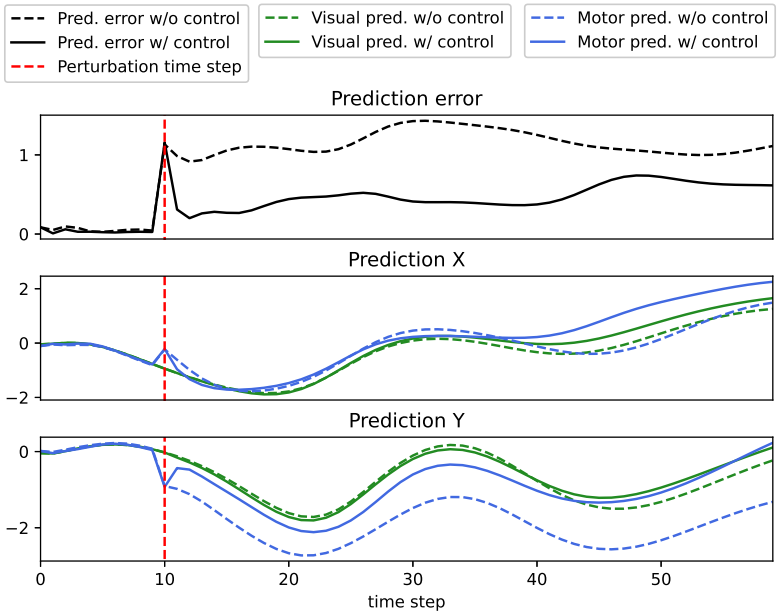


(a) Examples of trajectories obtained in the visual impairment experiment, for different impairment amplitudes $\sigma_i^2 \in \{0.0, 0.02, 0.04, 0.06, 0.08, 0.1\}$. Blue lines represent the uncontrolled motor trajectories displayed in the visual space. Green lines represent the visual trajectories predicted by the impaired visual RNN. Gray lines represent the corrected visual trajectories. These trajectories were obtained with an RNN hidden state dimension of 50 after training.

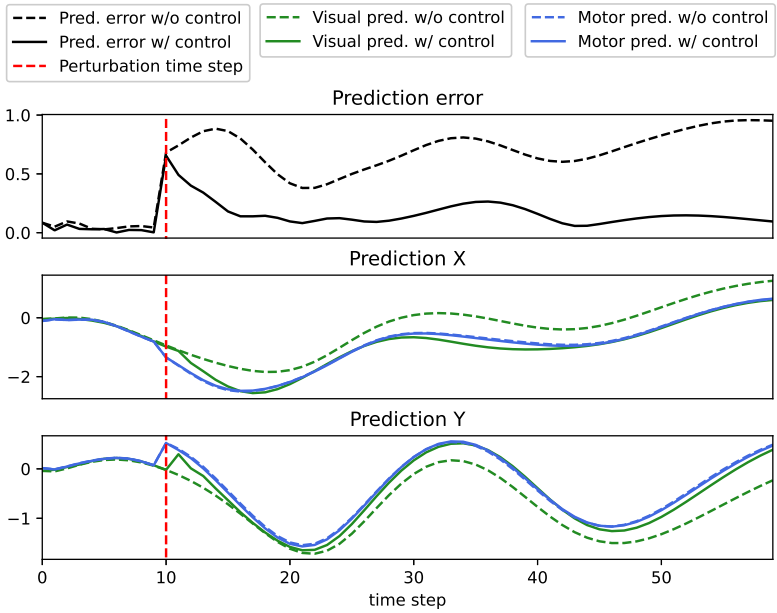


(b) Average prediction error according to the impairment amplitude σ_i^2. Prediction errors are measured as average distances with regard to corresponding test trajectories of the same label.

Figure 6.16: Impairments experiment. The visual RNN is impaired by applying a multiplicative noise of varying amplitude onto its parameters. Our model uses the motor prediction as a guide to correct the visual trajectory.



(a) Bidirectional influence favoring the visual to motor direction.



(b) Bidirectional influence favoring the motor to visual direction.

Figure 6.17: Bidirectional influence between the motor and visual models, with both feedback pathways activated at the same time. The top figures display the evolution of the prediction error $\|\boldsymbol{o}^v - \boldsymbol{o}^m\|_2^2$, while the middle and bottom figures display the evolution of the X and Y components of $\boldsymbol{o}^v$ and $\boldsymbol{o}^m$. Visual predictions $\boldsymbol{o}^v$ are represented in green, and motor predictions $\boldsymbol{o}^m$ are represented in blue. Dashed lines represent the trajectories generated by both models without control, i.e. when the interaction is turned off. Full lines represent the trajectories generated by both models with control.

In this section, we display results where we keep both error feedback pathways activated. We reuse the experimental set up from section 6.5.2.4, where we apply at time step $t = 10$ a random perturbation on the motor output of the model. These results are displayed in figure 6.17. The perturbation time step is marked by a red dashed line. We can see that the motor output is perturbed, which results in a mismatch between the motor (blue) and the visual (green) outputs.

Figure 6.17a displays a scenario where the bidirectional interaction favors influence from the visual model to the motor model. After the perturbation, the prediction error is promptly minimized due mostly to the correction of the motor model. The visual prediction slightly deviates from the trajectory it would exhibit in the absence of control (green dashed line).

On the contrary, 6.17b displays a scenario where the bidirectional interaction favors influence from the motor model to the visual model. After the perturbation, the prediction error is minimized mostly by the correction of the visual model. The motor prediction slightly deviates from the trajectory it would exhibit without control (blue dashed line).

These additional results prove that the model can implement a bidirectional influence where both feedback pathways are simultaneously activated.

6.5.3 Discussion

Here, we shortly discuss the biological plausibility of the proposed framework, as well as the possible directions of improvements.

From a neurocomputational point of view, several works advocate for the relevance of randomly connected RNNs as a computational model for cortical networks (Wang et al., 2010; Hoerzer et al., 2012; Mannella and Baldassarre, 2015). In particular, (Mannella and Baldassarre, 2015) suggest RC as a candidate approach to generate movements as neural trajectories in the motor cortex. However, the authors train these cortical networks through a supervised learning scheme, which would need target values in the motor space. A candidate learning mechanism working with indirect supervision is the Reward-Modulated Hebbian Learning proposed in (Hoerzer et al., 2012) and combined with supervised learning in the SUPERTREX architecture (Pyle and Rosenbaum, 2019). This mechanism uses a measure of success (similar to the reward) to weight the gradient updates of the reservoir readout weights. Instead, our approach relates to the internal model theory, suggesting that efferent copies of motor commands in the brain are provided as inputs to an internal forward model predicting the sensory outcomes of performed actions (Shadmehr et al., 2010). The interesting feature brought by AIF is that, in contrast with control theory where the heavy lifting is done by the inverse models, the reciprocal top-down and bottom-up information passing scheme allows inferring proper actions using an error signal between sensory predictions and predicted outcomes of actions. These types of internal models are thought to be encoded in the intraparietal sulcus and superior parietal lobule regions of the posterior parietal cortex, for reaching and grasping movements (Creem-Regehr, 2009), as well as drawing and handwriting (Planton et al., 2017).

On a higher level of abstraction, motor cognition (planning, decision-making) involves other brain structures such as the cerebellum and the prefrontal cortex, for the prediction of outcomes (Pezzulo and Cisek, 2016; Mushiake et al., 2006; Botvinick and An, 2009), and the basal ganglia, for the selection of action policies (Botvinick and An, 2009; Mannella and Baldassarre, 2015). In (Friston et al., 2016), AIF is proposed as a candidate model for goal-directed behavior relating to the brain structures listed above.

With this work, we have shown that it is possible to learn a sequence memory of motor trajectories without relying on direct supervision in the motor space. However, our results rely heavily on the fact that our architecture was endowed with a perfect forward model, and perfect gradient backpropagation through this forward model. The learning of such a forward model might in practice be very complicated, even more so in more complex environments. Our environment is very advantageous in the fact that sensory observations only depend on current actions, without any delayed influence. In a realistic environment, a proper model of how actions influence observations should take into account an inferred representation of the state of the environment. Adding to this complexity, in partially observable environments, the state representation must be inferred based on a history of observations each giving incomplete information about the true environmental state. A shortcoming of this work lies in these major simplifications.

6.6 Conclusion

In this chapter, we have seen that it is possible to plug in our sequence memory models onto architectures implementing AIF. The complete model is thus capable of jointly inferring which motor trajectories are best fit for a certain goal, and writing these trajectories in a sequence memory model. Contrary to the first proposed model, the second model we have presented is able to dynamically adjust the motor trajectories being read from the sequence memory. Additionally, we have shown that the obtained architecture presented a symmetry allowing information to propagate in both directions between the motor sequence memory and the visual sequence memory. While the first model needed extensive interaction with the environment to learn the adequate motor repertoire, the second model can learn a motor sequence memory using only internal simulations. Interaction with the environment is only needed to train the forward model predicting the sensory outcomes of actions.

While our results show that it is possible to learn motor sequence memories without any direct supervision in the motor space, the quality of the inferred motor sequences relies heavily on the used AIF method. In particular, the precision of the forward model and of the inverse propagation in the forward model plays an essential role in building the adequate motor trajectories in the second model we have proposed. Additionally, AIF methods still need to prove that they can learn complex behaviors using sparse supervision. For instance, the agent's preferences encoded in the biased generative model could only involve a modality that is constant most of the time. Observations rewarded under this biased generative model would be very sparse, since only a handful of observations would be considered as more probable than the others. Consequently, the agent would need to plan for a large number of steps to choose its action course. In contrast, our model had direct supervision at each time step, greatly simplifying the problem.

We can also discuss the choice of using an architecture using a forward model, instead of an architecture that jointly generates sensory observations and motor commands. Our intuition regarding this question is that generative models reproducing the causal structure of the world should perform better. Actions have an effect on the environment, which in turn provides observations. The causal structure of this interaction would thus place actions as parent causes directly influencing a random variable representing the state of the environment. Research in the fields of causality and machine learning seems to indicate that internal models reproducing the causal structure of the environment are more robust to changes in the environment or task definition (Ke et al., 2019; Bengio et al., 2019). More experiments comparing both approaches should be conducted to conclude on this topic.

In our AIF formulation, we did not discuss the possibility of modeling the distribution onto model parameters. This distribution can be taken into account in the VFE computation. We have seen that surprise directly depends on the precision of the recognition density onto the latent state variable. If the agent is certain of the latent state, it will be certain of the observation, and if the prediction is correct the resulting surprise would be lower than if it had done the same prediction with lower confidence. The degree of confidence in a prediction is thus weighted by the precision of the recognition distribution onto the latent state variable. Similarly, if we include model parameters in the Bayesian scheme, the precision of the recognition distribution onto the model parameters would influence the precision of the prediction. As such, minimizing surprise in the long-term could induce behaviors that favor learning.

We can go even further and try to include the graphical structure of the model as part of the inferable quantities. If the graphical structure of the model that is best suited for prediction and adaptability is indeed the one that reproduces the causal structure of the environment, then finding this structure could help to decrease future surprise. Crucially, it is known that to identify the true causal structure behind data, we have to rely on what is called *interventions*. Interventions are modifications of some variables in the true generative process (here, the environment), that cause changes in the observable variables. Without intervention, it is not possible to discriminate the true causal direction between two correlated variables (or if they have a mutual confounder). For instance, let us consider a data set containing data points indicating the recorded temperature inside a room, and whether the heating is turned on. We would observe in the data set a high correlation between the heating being turned on and the temperature. However, this is not enough to conclude about the causal relationship between the two variables. To this end, we could use interventions. For instance, we could try to increase the temperature of the room by other means (for instance by lighting a fire), and check whether this activates the heating or not. In

a hypothetical implementation of AIF capable of including these considerations, the agent could try to perform experiments in the form of interventions onto the environment in order to properly identify its true causal structure. Recent works in RL try to implement artificial curiosity-driven agents according to this principle (Perez et al., 2020; Sontakke et al., 2021).

At this point, we would start to have many layers of different drives influencing action selection: the extrinsic reward i.e. the desire to abide by the biased generative model, the motivation to seek out observations informative about the latent state, the motivation to seek out observations informative about the model parameters, and finally the motivation to perform interventions informative about the causal structure of the environment.

Finally, we can discuss the possibility of including several sensory modalities into our architecture. This would require learning forward models that associate motor commands with resulting observations in each modality. Based on such forward models, it would be possible to infer actions based on combined goals defined in these different modalities. Interestingly, some motor commands could be associated with high precision to resulting observations in one modality, but with low precision to observations in another modality. In this case, the precision coefficients would naturally weight the different modalities in order to infer actions based primarily on the modalities associated with higher precisions.

Chapter 7

Conclusion

7.1 Summary of the contributions

In this thesis, we have studied the question of sequence memory modeling using RNN architectures inspired by the PC theory. In chapter 3, we have designed several PC-based RNN models that can be trained as sequence memories. The proposed models differ in whether they make use of a hidden causes layer and first-order generalized coordinates. We have presented two comparative studies for sequence memory RNN models in different settings. The first study aimed at comparing the capacity of different sequence memory models using a data set of continuous 2D trajectories. In the second study, we have compared different learning algorithms for sequence memories in a continual learning setting, limiting ourselves to learning methods involving only temporally and spatially local information. These two comparative studies have highlighted some interests of the PC approach for sequence memory modeling: using hidden causes and first-order generalized coordinates seems to improve the memory capacity, and hidden causes also improve the performances of sequence memories in a continual learning setting.

An important question for sequence memory modeling is the design of memory retrieval methods. In chapter 5, we have proposed several iterative algorithms for memory retrieval. These algorithms make use of a top-down Gaussian mixture prior to attract the recall mechanism into desired values for the hidden causes variable. We have seen that the method exploiting a learned structure of the sequences written in the memory allowed us to properly retrieve the correct item even in the presence of noise or with only a fraction of the sequence available.

Finally, the sequence memory models presented in chapter 3, and the memory retrieval mechanisms presented in chapter 5, relied on the assumption that the sequences that should be learned (written) or retrieved (read) were directly available to the agent. We have shown in chapter 6 that we could relax this assumption and jointly construct and learn motor trajectories without direct supervision in the motor space, by building upon the AIF framework.

7.2 Discussion on the thesis choices

We can discuss some of the choices made to design our models.

- **Focusing on RNNs:** First, one of the first choices made during this thesis was to focus on recurrent neural architectures. While RNNs are naturally fit for sequential tasks, feedforward architectures such as CNNs or Transformers have also shown great results. Although we have presented these networks in our overview of sequence memory ANN models, we have not tried using them with PC-based inference and learning rules. Interestingly, since no recurrent dynamics are present in these models, we would avoid the problems we encountered by merging the RNN dynamics with the dynamics induced by PC. As such, inference in feedforward networks could be easier than in recurrent models. Still, feedforward architectures come with the disadvantage of scaling poorly to very long sequences and are not natural for modeling sequences of different lengths.
- **Number of inference steps:** Second, we can question whether the PC-RNN models should only perform one inference step for each new data point $\boldsymbol{x}_t^*$. It might be interesting to study

the possibilities offered by multiple inference steps for each observation, as it is done for instance with ERS-based inference (Ahmadi and Tani, 2017). For robotic implementations, this number of inference steps in peristimulus time could depend on the speed of the inference process and the frequency of the sensory sampling. Note that there should not be any interest in slowing down the sensory sampling in order to perform several inference steps per sample, since this would resume to performing one inference per sample but with outdated information. We have ignored this question in our implementations, but this could be a straightforward way of improving our models.

- **Learning method:** Finally, in many experiments we chose to use the BPTT learning rules on our models, instead of the PC-based learning rules that we have derived. Our experiments with both methods show that BPTT largely outperforms PC-based learning rules. The main interest of PC-based learning rules lies in that they can be performed online, using only information locally available. We have tried to highlight this interest in our second comparative study, but we believe that more work should be done in order to assess the advantages of PC-based learning rules. In particular, these learning rules could be naturally implemented on dedicated neuromorphic hardware reproducing the connectivity patterns proposed by PC.

7.3 Limitations

In this section, we discuss some limitations of the works presented in this thesis.

- **Lack of complexity in our experiments:** An important and valid criticism about our work is the simplicity of the task on which we have applied our models. Throughout this thesis, we have experimented with sequence memories of 2D continuous trajectories. However, sequence memory models could in principle be used to store highly discontinuous data in a very large space. We do not know if the proposed models would scale properly to more complex tasks like storing sequences of images or words. Still, the goal of this thesis was to conduct a horizontal research on the topic of sequence memories with PC, rather than an in-depth experimentation with sequence memory models on different applications. Choosing a simple applicative setting allowed us to explore more questions such as memory retrieval and motor sequence memory learning.

- **Supervision:** We have always considered sequence memory training as a supervised learning problem, where both the key and the associated sequence are available. In this regard, we have made the assumption that the agent always knew the index (or label, or category) of the trajectory it was being shown. In real-world settings, this information might not be available to the agent. We can consider two situations. In the first situation, the agent is being shown a trajectory that it has already seen. In this case, we can suppose that the memory retrieval process would be able to recognize the sequential pattern and pursue learning with the newly available information. In the second situation, the agent is being shown a trajectory that it has never seen. The detection of this distributional shift might be trickier using our models, but for instance, could be based on a threshold recognition time. If after this set amount of time, the new trajectory is not recognized, the agent could create a new mode in its generative model in order to account for this new sequential pattern. Still, we have not experimented with this situation and always assumed that this information was available to the learning system.

- **Limitations of PC for the inference of discrete variables:** An important limitation of the PC approach was observed on the question of memory retrieval. As we have seen, the PC theory can be derived from the FEP, as a variational Bayes method resting on some additional approximations. Especially, we have assumed that the recognition density is a unimodal distribution. Minimizing VFE through gradient descent might work properly in the case of unimodal distributions, but it also limits the inference process as it is not capable of considering several possible values at the same time. Consequently, the inference process is not enough to infer discrete latent variables as in the memory retrieval problem and get stuck in local basins of VFE. To work around this issue, we had to inject some noise into the inference process. Still, we believe that this issue constitutes an important limitation of PC

and that other variational inference methods not relying on this unimodal assumption might be of interest (for instance using statistical ensembles/particle filters (Friston, 2008b)).

- **Simplifications in the AIF experiments:** Finally, we shortly summarize the limitations of the motor sequence memory models that we have highlighted in chapter 6. Our results relied heavily on the simplicity of the task at hand, and on the assumption that a perfect forward model was available for the agent. Experiments in more complex environments, with an agent that jointly learns an internal model of the environment and a repertoire of motor trajectories, are needed to better evaluate the general architecture we have proposed.

7.4 Recommendations for future work

To conclude, we describe future research directions that we deem of interest regarding sequence memory modeling.

- We believe that an important benefit of the proposed models comes from their biological inspiration. Since the PC-based inference and learning algorithms only involve temporally and spatially local information, they could very well be implemented on dedicated neuromorphic hardware reproducing the connection patterns involved in PC. This might potentially allow to train very large PC-based models at a low energy cost. Dedicated hardware for other alternatives to BP such as Direct Feedback Alignment (Nøkland, 2016) are already being investigated (Launay et al., 2020).

- The concept of precision weighting brought by PC has been used in our models, but the precision coefficients were always uniform inside a same layer, and manually set by the experimenter. We have seen that these coefficients play an immense role in the behavior of our models:

  - They weight the bottom-up and top-down signals in our networks' update rules. As such, the latent representations reached by in the intermediate layers can be seen as weighted agreements between bottom-up and top-down forces. Especially, the bottom-up update of the hidden layer is weighted by the precision of the observation. Using adequate precision coefficients can thus help our models properly handle noisy observations by not overshooting during inference.
  - Since they weight the bottom-up transport of prediction error information, they directly influence the learning rate applied at each layer during the Hebbian update of the synaptic weights. It could be very interesting to study the effect of different precision coefficients choices onto the evolution of the synaptic weights on the different layers.
  - We have seen that these coefficients can direct the influence between the motor and perceptual models in our bidirectional architecture. Information is primarily transported from the model with the highest precision towards the model with the lowest precision.
  - Using distinct precision coefficients for each dimension of the same variable can endow the model with attention mechanisms (Feldman and Friston, 2010), the precision coefficients weighting how much a certain dimension of the prediction error signal should be propagated to higher levels. This improved mechanism would allow to disregard the prediction error coming from very noisy variables for which the precision would be low. Additionally, if the precision coefficients were to be actively estimated by the agent, they could be used to ignore observed information deemed irrelevant by the agent.

  Future research on this subject could study the effect of different precision coefficients estimation methods onto the different mechanisms listed above.

- Another natural research direction would be to study hierarchical architectures using our models. The question of hierarchy has been left out even though it is an important property necessary to develop complex perceptual capabilities and complex decision-making skills. In particular, increasing the depth in our models can give rise to vanishing gradient phenomena. Whether the PC-based inference and learning mechanisms can work around this issue is still an open question. Another open question would be about the interest of using a hidden causes layer at each level of the hierarchy.

- Finally, we believe that it could be interesting to study the formation of the structure of the hidden causes variables in the sequence memory. We have seen in chapter 5 that a structured hidden causes space yielded faster retrieval processes. For instance, could these mechanisms benefit from a more tree-like structure to perform an organized research into the repertoire of known trajectories?

Appendices

A Variational free-energy simplifications

In this appendix, we provide more detailed derivations of the VFE expression given in chapter 2. We start from the definition of VFE, denoted F:

$$F(\boldsymbol{x}) = \int_h \log\Big(\frac{q(\boldsymbol{h})}{p(\boldsymbol{x},\boldsymbol{h})}\Big) q(\boldsymbol{h}) d\boldsymbol{h} \tag{1}$$

where $\boldsymbol{X}$ is an observed random variable and $\boldsymbol{H}$ is a latent random variable. The probability model p corresponds to the agent's generative model of its observations, while the distribution q is the recognition density, representing the agent's current belief about the latent variable. As our goal here is to show how variational inference aligns with PC, we start from another way of writing down VFE.

$$F(\boldsymbol{x}) = \underbrace{\int_h E(\boldsymbol{x},\boldsymbol{h}) q(\boldsymbol{h}) d\boldsymbol{h}}_{\text{Expected energy}} + \underbrace{\int_h \log\big(q(\boldsymbol{h})\big) q(\boldsymbol{h}) d\boldsymbol{h}}_{\text{(Negative) entropy}} \tag{2}$$

where

$$E(\boldsymbol{x},\boldsymbol{h}) = -\log p(\boldsymbol{x},\boldsymbol{h}) \tag{3}$$

is called *energy*. The first term of this equation thus corresponds the expectation of energy with regard to the density q. The second term is the entropy of the recognition density q.

In this framework, perceptual inference is framed as a process of free-energy minimization by optimizing the recognition density q. In practice, q is constrained to certain classes of probability distributions to simplify the optimization problem. Here we assume that the recognition density takes a Gaussian form, and VFE is minimized by varying the parameters of this distribution.

$$q(\boldsymbol{h}) = \frac{1}{\sqrt{(2\pi)^{d_h}|\boldsymbol{\Sigma}|}} \exp\Big(-\frac{1}{2}(\boldsymbol{h}-\boldsymbol{m}_h)^\intercal \cdot \boldsymbol{\Sigma}^{-1} \cdot (\boldsymbol{h}-\boldsymbol{m}_h)\Big) \tag{4}$$

where d_h is the dimension of H, and $\boldsymbol{m}_h$ and $\boldsymbol{\Sigma}$ are the mean vector and covariance matrix of the multivariate Gaussian distribution on H. $|\cdot|$ denotes the determinant of a matrix. Integrating this definition into equation 2 gives us:

$$\begin{aligned} F(\boldsymbol{x}) &= -\frac{1}{2}\log\big((2\pi)^{d_h}|\boldsymbol{\Sigma}|\big) - \int_h \big(\frac{1}{2}(\boldsymbol{h}-\boldsymbol{m}_h)^\intercal \cdot \boldsymbol{\Sigma}^{-1} \cdot (\boldsymbol{h}-\boldsymbol{m}_h)\big) q(\boldsymbol{h}) d\boldsymbol{h} + \int_h E(\boldsymbol{x},\boldsymbol{h}) q(\boldsymbol{h}) d\boldsymbol{h} \\ &= -\frac{1}{2}\log\big((2\pi)^{d_h}|\boldsymbol{\Sigma}|\big) - \frac{d_h}{2} + \int_h E(\boldsymbol{x},\boldsymbol{h}) q(\boldsymbol{h}) d\boldsymbol{h} \end{aligned} \tag{5}$$

The first two terms of this equation do not depend on the recognition density mean $\boldsymbol{m}_h$. The last term however, needs some approximation to be computed. As explained before, variational inference draws its interest from the fact that expectations with regard to $q(\boldsymbol{h})$ are easier to compute than expectations with regard to $p(\boldsymbol{h})$. With our Gaussian assumption, this is only the case if we suppose that $q(\boldsymbol{h})$ is tightly peaked around its mean $\boldsymbol{m}_h$. With this additional assumption, we can use the Taylor expansion of $E(\boldsymbol{x},\boldsymbol{h})$ around the value $\boldsymbol{h} = \boldsymbol{m}_h$:

$$E(\boldsymbol{x},\boldsymbol{h}) \approx E(\boldsymbol{x},\boldsymbol{m}_h) + \big(\nabla_{\boldsymbol{m}_h} E(\boldsymbol{x},\boldsymbol{m}_h)\big) \cdot (\boldsymbol{h}-\boldsymbol{m}_h) + \frac{1}{2}(\boldsymbol{h}-\boldsymbol{m}_h)^\intercal \cdot \boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)} \cdot (\boldsymbol{h}-\boldsymbol{m}_h) \tag{6}$$

where $\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)}$ is the Hessian matrix containing in cell (i,j) the derivative $\frac{\partial^2 E(\boldsymbol{x},\boldsymbol{m}_h)}{\partial m_{h,i}\partial m_{h,j}}$. When computing the expectation of $E(\boldsymbol{x},\boldsymbol{h})$ with regard to $q(\boldsymbol{h})$, the first-order term of the expansion is canceled out (the expectation of $(\boldsymbol{h}-\boldsymbol{m}_h)$ is 0), and in the second-order term of the expansion appears the covariance matrix $\boldsymbol{\Sigma}$:

$$\int_h E(\boldsymbol{x},\boldsymbol{h})q(\boldsymbol{h})d\boldsymbol{h} \approx E(\boldsymbol{x},\boldsymbol{m}_h) + \frac{1}{2}\mathrm{Tr}\big(\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)} \cdot \boldsymbol{\Sigma}\big) \tag{7}$$

where Tr denotes the trace of a matrix. Using all these assumptions, we can rewrite the VFE as:

$$F(\boldsymbol{x}) \approx E(\boldsymbol{x},\boldsymbol{m}_h) + \frac{1}{2}\Big(\mathrm{Tr}\big(\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)} \cdot \boldsymbol{\Sigma}\big) - \log\big((2\pi)^{d_h}|\boldsymbol{\Sigma}|\big) - d_h\Big) \tag{8}$$

To obtain the final approximation of the VFE, we now assume that the recognition density covariance matrix is proportional to the identity matrix $\mathbb{I}_{d_h}$:

$$\boldsymbol{\Sigma} = \zeta\mathbb{I}_{d_h} \tag{9}$$

This assumption greatly simplifies the second term of the VFE. Intuitively, this simply means that the recognition density assumes that all components of the latent state H are independent and estimates them with the same degree of confidence. Finally, we obtain the following expression:

$$F(\boldsymbol{x}) \approx E(\boldsymbol{x},\boldsymbol{m}_h) + \frac{1}{2}\Big(\zeta\mathrm{Tr}\big(\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)}\big) - d_h\log(2\pi\zeta) - d_h\Big) \tag{10}$$

We can notice that only the second term of this equation depends on ζ. Since our goal is to find the optimal values for $\boldsymbol{m}_h$ and ζ, we can already find a closed-form solution for ζ.

$$\frac{\partial F}{\partial \zeta} = \frac{1}{2}\Big(\mathrm{Tr}\big(\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)}\big) - \frac{d_h}{\zeta}\Big) \tag{11}$$

This derivative falls to 0 when:

$$\zeta = \zeta^* = \frac{d_h}{\mathrm{Tr}\big(\boldsymbol{H}_{E(\boldsymbol{x},\boldsymbol{m}_h)}\big)} \tag{12}$$

If we inject this solution ζ into the expression of the VFE, we obtain:

$$F(\boldsymbol{x}) \approx E(\boldsymbol{x},\boldsymbol{m}_h) - \frac{d_h}{2}\log(2\pi\zeta^*) \tag{13}$$

We have reached the end of the derivations of the VFE using our set of assumptions. This process has cast an expression involving integrals into a function depending only on the observed variable $\boldsymbol{x}$ and the recognition density mean $\boldsymbol{m}_h$ (note that ζ^* is completely characterized by these two variables). Perceptual inference using these hypotheses results in a process of iterative optimization of $\boldsymbol{m}_h$ in order to decrease the quantity $E(\boldsymbol{x},\boldsymbol{m}_h)$. Therefore, in the upcoming derivations, we simply consider the optimization of $E(\boldsymbol{x},\boldsymbol{m}_h)$ as a proxy for $F(\boldsymbol{x})$.

To obtain an algorithm that relates to the PC theory, we further need to make assumptions on the generative model $p(\boldsymbol{x},\boldsymbol{h})$. First, we assume that the generative model is hierarchical and can be factored as:

$$p(\boldsymbol{x},\boldsymbol{h}) = p(\boldsymbol{x}|\boldsymbol{h}^{(1)})p(\boldsymbol{h}^{(1)}|\boldsymbol{h}^{(2)})\dots p(\boldsymbol{h}^{(l-1)}|\boldsymbol{h}^{(l)})p(\boldsymbol{h}^{(l)}) \tag{14}$$

with $\boldsymbol{h}$ composed of l random variables $(\boldsymbol{h}^{(1)}\cdots\boldsymbol{h}^{(l)})$. The probabilistic model does not necessarily need to be a chain. We could make each variable depend on its parents in any hierarchical probabilistic model. For simplicity, we keep the chain hypothesis in the factorization of this probabilistic model.

Second, we suppose that each conditional probability is a multivariate Gaussian:

$$p(\boldsymbol{x}|\boldsymbol{h}^{(1)}) = \frac{1}{\sqrt{(2\pi)^{d_x}|\boldsymbol{\Sigma}_0|}} \exp\Big(-\frac{1}{2}\big(\boldsymbol{x}-\boldsymbol{g}^{(1)}(\boldsymbol{h}^{(1)})\big)^\intercal \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \big(\boldsymbol{x}-\boldsymbol{g}^{(1)}(\boldsymbol{h}^{(1)})\big)\Big) \tag{15}$$

$$p(\boldsymbol{h}^{(i)}|\boldsymbol{h}^{(i+1)}) = \frac{1}{\sqrt{(2\pi)^{d_{h^{(i)}}}|\boldsymbol{\Sigma}_i|}} \exp\Big(-\frac{1}{2}\big(\boldsymbol{h}^{(i)}-\boldsymbol{g}^{(i+1)}(\boldsymbol{h}^{(i+1)})\big)^\intercal \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \big(\boldsymbol{h}^{(i)}-\boldsymbol{g}^{(i+1)}(\boldsymbol{h}^{(i+1)})\big)\Big) \tag{16}$$

$$p(\boldsymbol{h}^{(l)}) = \frac{1}{\sqrt{(2\pi)^{d_{h^{(l)}}}|\boldsymbol{\Sigma}_l|}} \exp\Big(-\frac{1}{2}\big(\boldsymbol{h}^{(l)}-\boldsymbol{\mu}\big)^\intercal \cdot \boldsymbol{\Sigma}_l^{-1} \cdot \big(\boldsymbol{h}^{(l)}-\boldsymbol{\mu}\big)\Big) \tag{17}$$

The parameters of the generative model are the collections of covariance matrices $\boldsymbol{\Sigma}_i$, for $0 \leq i \leq l$, and the prior mean $\boldsymbol{\mu}$. The generative model is hierarchical in that the mean of the Gaussian density at layer (i) depends on the state $\boldsymbol{h}^{(i+1)}$ of the upper layer, through a function $\boldsymbol{g}^{(i+1)}$. Using the PC vocabulary, these means are called the predictions of the generative model, and $\boldsymbol{\mu}$, $\boldsymbol{\Sigma}_i$ as well as the parameters $\boldsymbol{\theta}^{(i)}$ of the functions $\boldsymbol{g}^{(i)}$, are called the parameters of the generative model.

Based on this definition of the generative model, we can derive an expression of the variational-free-energy for equation 13:

$$\begin{aligned} F(\boldsymbol{x}, \boldsymbol{m}_h^{(1)}, \dots, \boldsymbol{m}_h^{(l)}) =& E(\boldsymbol{x}, \boldsymbol{m}_h^{(1)}, \dots, \boldsymbol{m}_h^{(l)}) + C \\ =& -\log p(\boldsymbol{x}|\boldsymbol{m}_h^{(1)}) \\ & - \sum_{i=1}^{l-1} \log p(\boldsymbol{m}_h^{(i)}|\boldsymbol{m}_h^{(i+1)}) \\ & - \log p(\boldsymbol{m}_h^{(l)}) \\ & + C \end{aligned} \tag{18}$$

where C corresponds to the second term of F in equation 13, that does not depend on the observation $\boldsymbol{x}$ nor on the recognition density means. This gives us:

$$\begin{aligned} F(\boldsymbol{x}, \boldsymbol{m}_h^{(1)}, \dots, \boldsymbol{m}_h^{(l)}) =& \frac{1}{2}\big(\boldsymbol{x}-\boldsymbol{g}^{(1)}(\boldsymbol{m}_h^{(1)})\big)^\intercal \cdot \boldsymbol{\Sigma}_0^{-1} \cdot \big(\boldsymbol{x}-\boldsymbol{g}^{(1)}(\boldsymbol{m}_h^{(1)})\big) \\ & + \frac{1}{2}\sum_{i=1}^{l-1} \big(\boldsymbol{m}_h^{(i)}-\boldsymbol{g}^{(i+1)}(\boldsymbol{m}_h^{(i+1)})\big)^\intercal \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \big(\boldsymbol{m}_h^{(i)}-\boldsymbol{g}^{(i+1)}(\boldsymbol{m}_h^{(i+1)})\big) \\ & + \frac{1}{2}\big(\boldsymbol{m}_h^{(l)}-\boldsymbol{\mu}\big)^\intercal \cdot \boldsymbol{\Sigma}_l^{-1} \cdot \big(\boldsymbol{m}_h^{(l)}-\boldsymbol{\mu}\big) \\ & + C' \end{aligned} \tag{19}$$

where we have included in C' the previous constant C, as well as new terms only depending on the covariance matrices $\boldsymbol{\Sigma}_i$. We have kept in this expression only the terms that depend on the recognition density means $\boldsymbol{m}_h^{(i)}$. We can define prediction error $\boldsymbol{\epsilon}^{(i)}$ at each layer, that measures the shift from the prediction coming from the upper layer, and the current recognition density mean:

$$\begin{aligned} \boldsymbol{\epsilon}^{(0)} &= x - \boldsymbol{g}^{(1)}(\boldsymbol{m}_h^{(1)}) \\ \boldsymbol{\epsilon}^{(i)} &= \boldsymbol{m}_h^{(i)} - \boldsymbol{g}^{(i+1)}(\boldsymbol{m}_h^{(i+1)}) \\ \boldsymbol{\epsilon}^{(l)} &= \boldsymbol{m}_h^{(l)} - \boldsymbol{\mu} \end{aligned} \tag{20}$$

The definition of prediction errors has the advantage of simplifying the expression of the VFE, but also are meaningful when transcribing our inference algorithm into a neural network architecture. We obtain the final expression of VFE based on all the previous assumptions:

$$F(\boldsymbol{x}, \boldsymbol{m}_h^{(1)}, \dots, \boldsymbol{m}_h^{(l)}) = \frac{1}{2}\sum_{i=0}^{l} \boldsymbol{\epsilon}^{(i)\intercal} \cdot \boldsymbol{\Sigma}_i^{-1} \cdot \boldsymbol{\epsilon}^{(i)} + C' \tag{21}$$

B Hyperparameters for the continual learning benchmark

Model	Parameters
Parameters common to all models that are not optimized	$d_h = 300$ $d_x = 2$ $d_c = 20$ (when relevant) $\alpha_x = 10^{-5}$ (when relevant) $\alpha_h = 0$ (when relevant) $\tau = 50$ (when relevant)
ESN	$\lambda = 3, 3.10^{-6}$
Conceptors	$\lambda = 8, 9.10^{-5}$ $\alpha = 1, 2$
PC-RNN-V	$\lambda = 1, 9.10^{-6}$ $\lambda_r = 53, 4$
P-TNCN	$\lambda = 3, 7.10^{-6}$ $\lambda_b = 3, 7.10^{-6}$ $\lambda_r = 0, 01$
PC-RNN-Hebb	$\lambda = 3, 4.10^{-6}$ $\lambda_b = 3, 4.10^{-6}$ $\lambda_r = 0, 04$
PC-RNN-HC-A	$\lambda = 8, 3.10^{-6}$ $\lambda_r = 0, 1$ $\lambda_c = 2.10^5$
PC-RNN-HC-M	$\lambda = 6, 3.10^{-5}$ $\lambda_r = 3, 1$ $\lambda_c = 1, 1.10^5$
PC-RNN-HC-A-RS	$\lambda = 6, 8.10^{-6}$ $\lambda_r = 4, 4$ $\sigma = 6, 0.10^{-5}$
PC-RNN-HC-M-RS	$\lambda = 6.10^{-6}$ $\lambda_r = 1, 1$ $\sigma = 1, 0.10^{-5}$
PC-RNN-HC-A + Conceptors	$\lambda = 8, 9.10^{-5}$ $\lambda_r = 0, 1$ $\lambda_c = 1, 2.10^5$ $\alpha = 6, 2$

Table 1: Hyperparameters for the continual learning benchmark.

C Continual learning figures

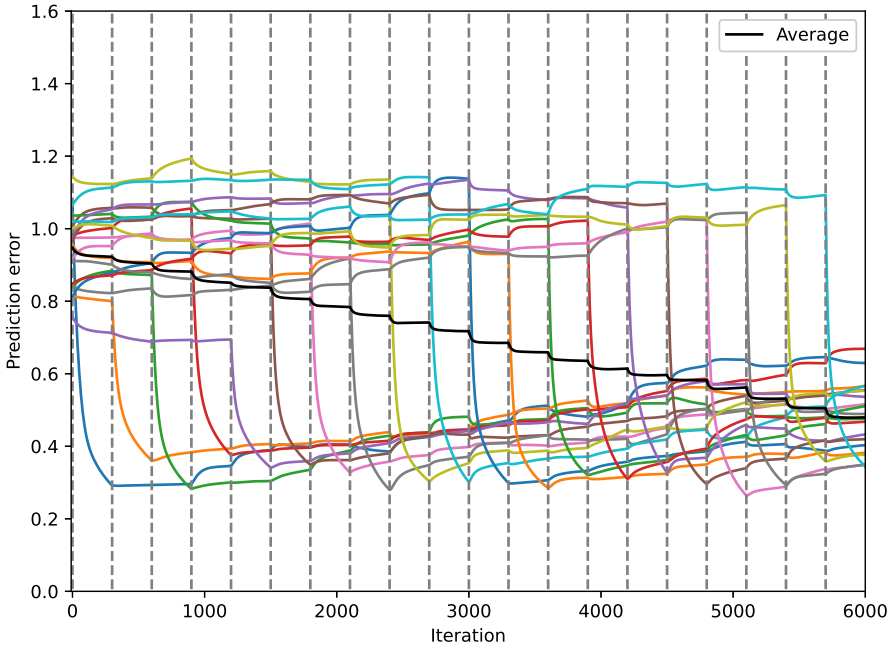


Figure 1: Continual learning results with the PC-RNN-V model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-V model.

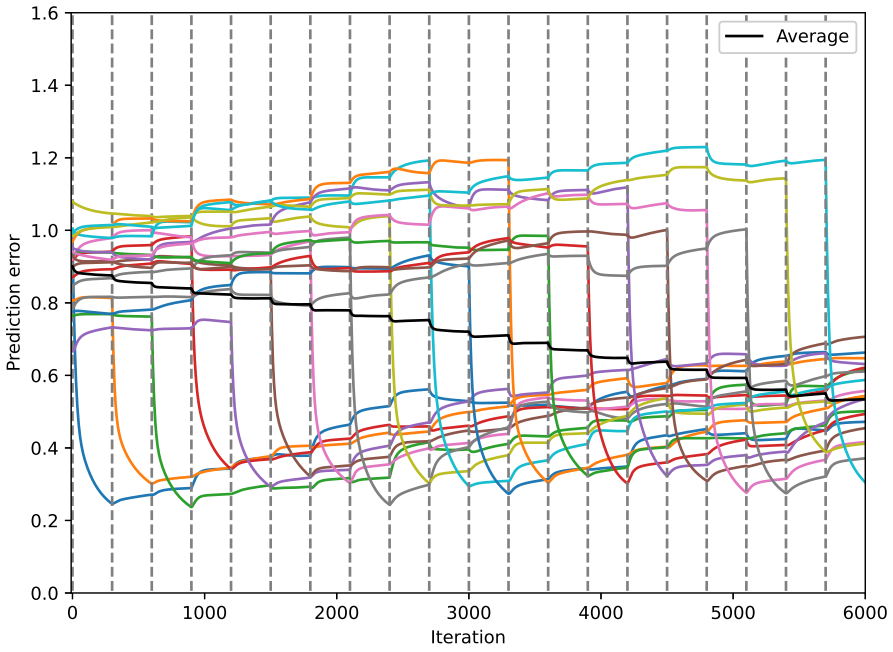


Figure 2: Continual learning results with the P-TNCN model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the P-TNCN model.

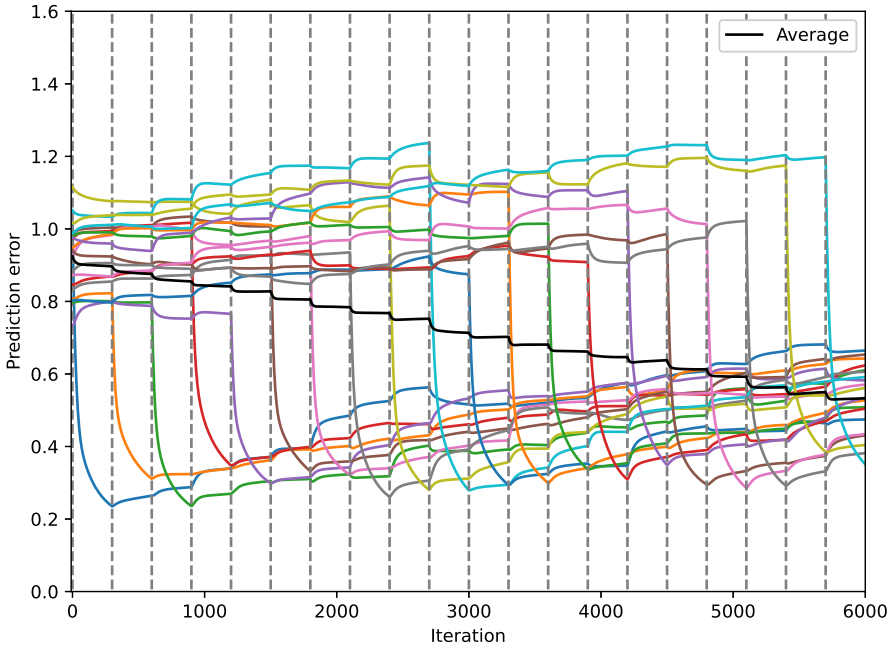


Figure 3: Continual learning results with the PC-RNN-Hebb model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the Hebbian learning variant of the PC-RNN-V model.

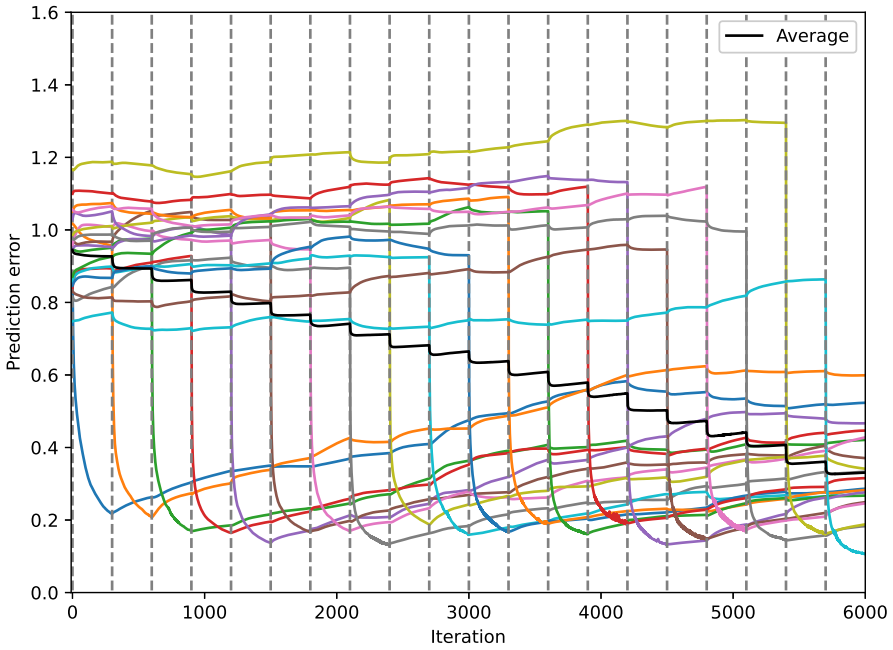


Figure 4: Continual learning results with the PC-RNN-HC-A model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-HC-A model.

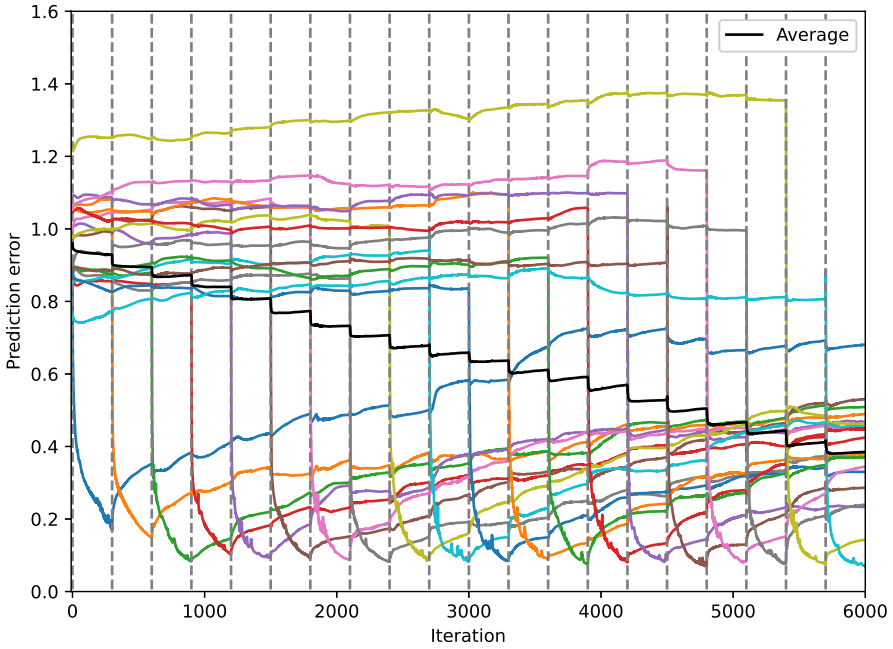


Figure 5: Continual learning results with the PC-RNN-HC-M model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-HC-M model.

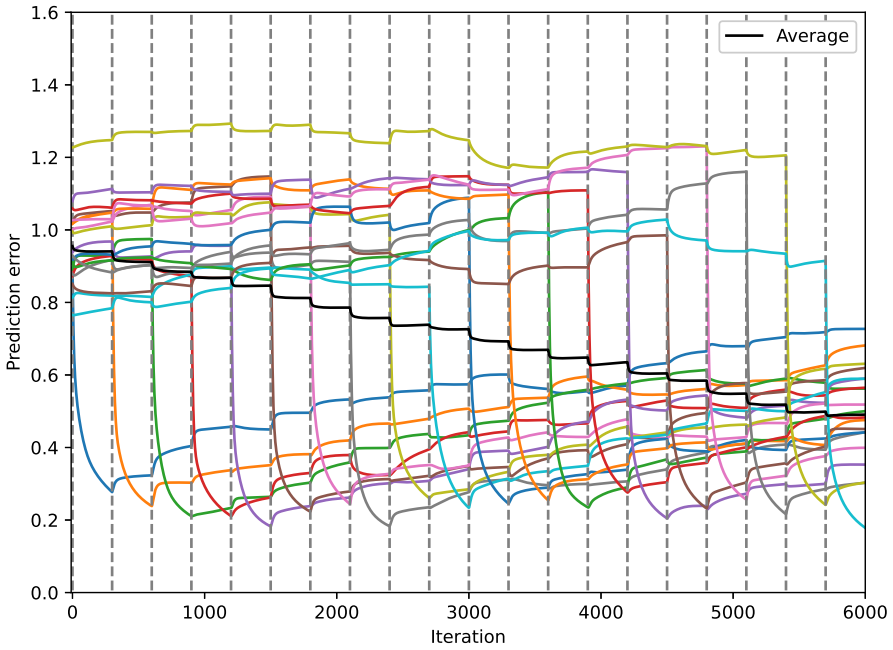


Figure 6: Continual learning results with the PC-RNN-A-RS model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-A model trained with random search.

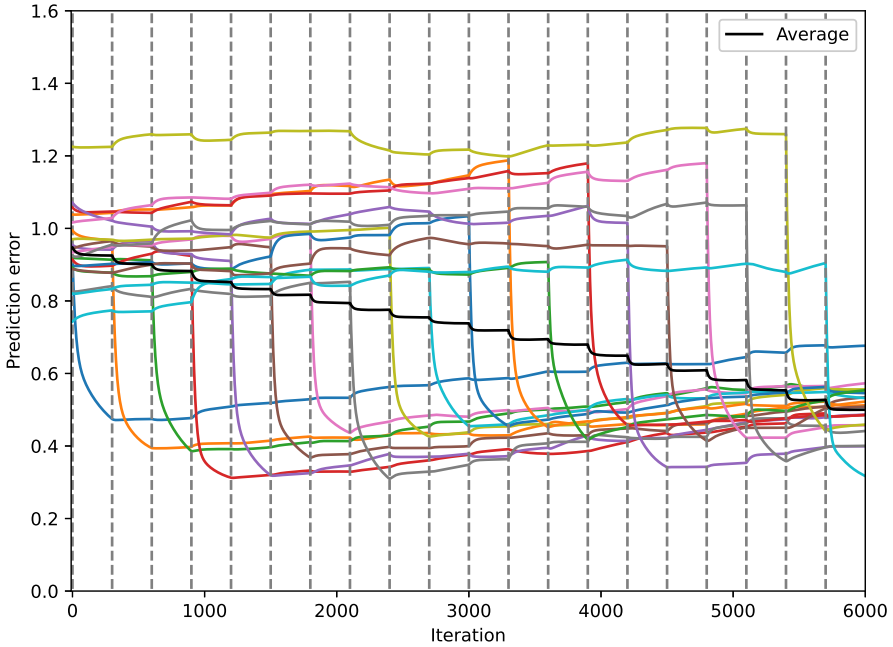


Figure 7: Continual learning results with the PC-RNN-M-RS model. We represent the average prediction error over 10 seeds, for the continual learning of 20 sequential patterns, using the PC-RNN-M model trained with random search.

D Distribution of the convergence times during memory retrieval

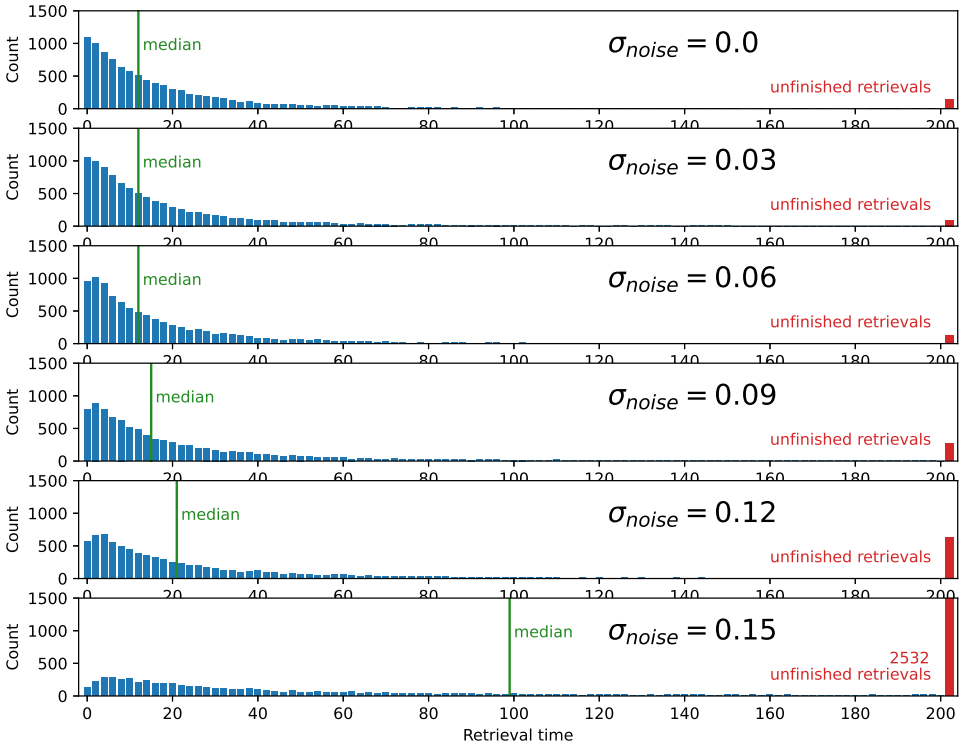


Figure 8: Distribution of the memory retrieval time according to the noise amplitude. The median values are represented by vertical green lines. The memory retrieval attempts that did not converge in the allowed time are shown in red.

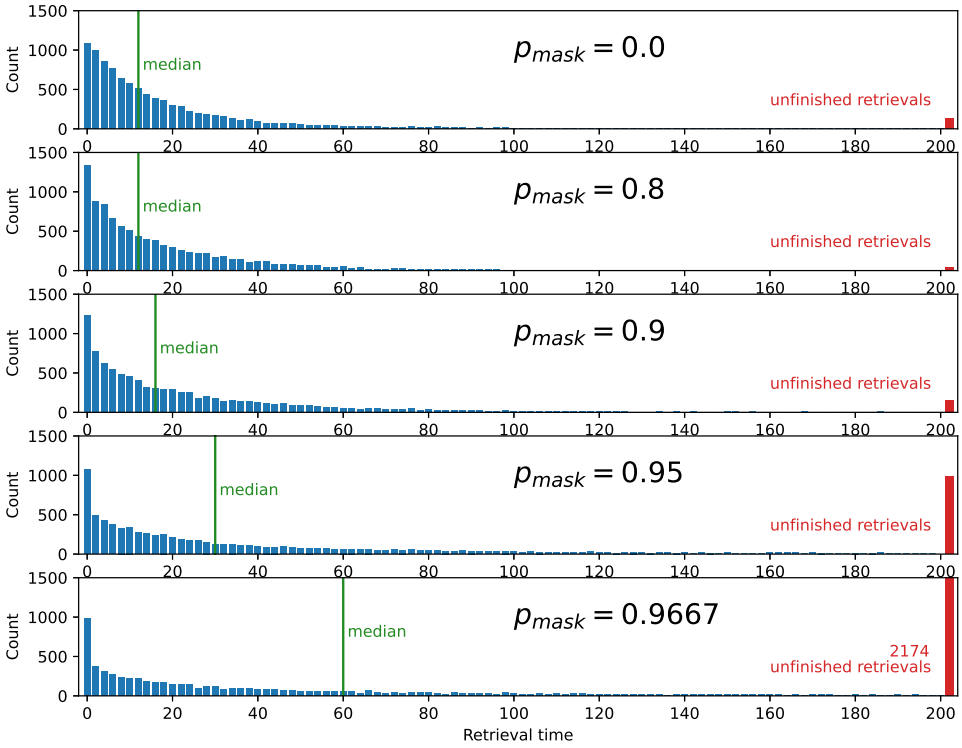


Figure 9: Distribution of the memory retrieval time according to the ratio of masked information. The median values are represented by vertical green lines. The memory retrieval attempts that did not converge in the allowed time are shown in red.

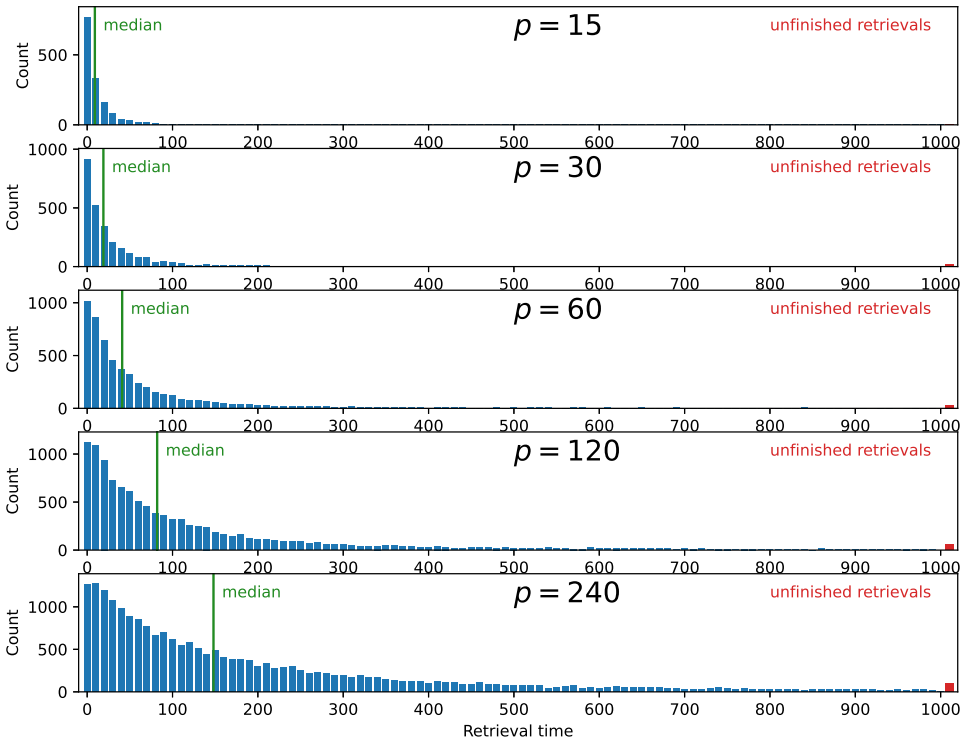


Figure 10: Distribution of the memory retrieval time according to the sequence memory size. The median values are represented by vertical green lines. The memory retrieval attempts that did not converge in the allowed time are shown in red.

E Parameters used in the motor sequence memory experiments

We provide the parameter values used for each experiment:

Name	Sections	Figures	Parameters
Parameters common to all experiments unless stated otherwise	-	-	$\dim(\mathbf{h}^v)=\dim(\mathbf{h}^m)=n=50$ $\dim(\mathbf{c}^v)=\dim(\mathbf{c}^m)=p=3$ $\dim(\mathbf{o})=2$ $\dim(\mathbf{m})=3$ $\tau = 7$
Visual RNN training	6.5.2.1	-	$\alpha_h^v = 0.001$ $\alpha_c^v = 0$ $\lambda_{out}^v = 0.1 \times 2^{-i/1000}$ $\lambda_f^v = 3$ $\lambda_p^v = 3$ $\lambda_c^v = 3$
Motor RNN training	6.5.2.1	-	$\alpha_m = 5$ $\alpha_h^m = 0.0001$ $\alpha_c^m = 0$ $\lambda_{out}^m = 0.3$ $\lambda_f^m = 100$ $\lambda_p^m = 100$ $\lambda_v^m = 100$
Motor RNN prediction	6.5.2.1 6.5.2.2	Figure 6.8 Figure 6.9	$\alpha_m = 0$ $\alpha_h^m = 0$ $\alpha_c^m = 0$
Intermittent motor control	6.5.2.3	Figure 6.10	$\alpha_m = 0.1$ $\alpha_h^m = 0.1$ $\alpha_c^m = 0$
Motor adaptation to perturbations	6.5.2.4	Figure 6.11	$\alpha_m = 0.15$ $\alpha_h^m = 0.1$ $\alpha_c^m = 0$
Motor adaptation to transformed visual targets	6.5.2.5 6.5.2.6	Figure 6.12 Figure 6.13 Figure 6.14	$\alpha_m = 4$ $\alpha_h^m = 0.15$ $\alpha_c^m = 0$
Visual adaptation with impairments	6.5.2.7	Figure 6.16	$\alpha_h^v = 0.05$ $\alpha_c^v = 0$
Bidirectional interaction visual to motor	6.5.2.8	Figure 6.17a	$\alpha_m = 0.1$ $\alpha_h^m = 0.05$ $\alpha_c^m = 0.1$ $\alpha_h^v = 0.01$ $\alpha_c^v = 0.1$
Bidirectional interaction motor to visual	6.5.2.8	Figure 6.17b	$\alpha_m = 0.03$ $\alpha_h^m = 0.01$ $\alpha_c^m = 0.1$ $\alpha_h^v = 0.1$ $\alpha_c^v = 0.3$

Table 2: Parameters used in the motor sequence memory experiments.

Bibliography

Ahmadi, Ahmadreza and Jun Tani (2017). "How can a recurrent neurodynamic predictive coding model cope with fluctuation in temporal patterns? Robotic experiments on imitative interaction". In: *Neural Networks* 92. Advances in Cognitive Engineering Using Neural Networks, pp. 3–16. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2017.02.015.

— (Nov. 2019). "A Novel Predictive-Coding-Inspired Variational RNN Model for Online Prediction and Recognition". In: *Neural Computation* 31.11, pp. 2025–2074. ISSN: 0899-7667. DOI: 10.1162/neco_a_01228. eprint: https://direct.mit.edu/neco/article-pdf/31/11/2025/1864800/neco_a_01228.pdf. URL: https://doi.org/10.1162/neco%5C_a%5C_01228.

Alink, Arjen, Caspar M. Schwiedrzik, Axel Kohler, Wolf Singer, and Lars Muckli (2010). "Stimulus Predictability Reduces Responses in Primary Visual Cortex". In: *Journal of Neuroscience* 30.8, pp. 2960–2966. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.3730-10.2010. eprint: https://www.jneurosci.org/content/30/8/2960.full.pdf. URL: https://www.jneurosci.org/content/30/8/2960.

Annabi, Louis, Alexandre Pitti, and Mathias Quoy (2020). "Autonomous learning and chaining of motor primitives using the Free Energy Principle". In: *2020 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8. DOI: 10.1109/IJCNN48605.2020.9206699.

— (2021a). "A Predictive Coding Account for Chaotic Itinerancy". In: *Artificial Neural Networks and Machine Learning – ICANN 2021*. Ed. by Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter. Cham: Springer International Publishing, pp. 581–592. ISBN: 978-3-030-86362-3.

— (2021b). "Bidirectional interaction between visual and motor generative models using Predictive Coding and Active Inference". In: *Neural Networks*. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.07.016. URL: https://www.sciencedirect.com/science/article/pii/S0893608021002793.

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473.

Baker, Bowen, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch (2019). "Emergent tool use from multi-agent autocurricula". In: *arXiv preprint arXiv:1909.07528*.

Barto, Andrew, Marco Mirolli, and Gianluca Baldassarre (2013). "Novelty or Surprise?" In: *Frontiers in Psychology* 4, p. 907. ISSN: 1664-1078. DOI: 10.3389/fpsyg.2013.00907.

Bayer, Justin and Christian Osendorfer (2015). *Learning Stochastic Recurrent Networks*. arXiv: 1411.7610 [stat.ML].

Bengio, Y., P. Simard, and P. Frasconi (Mar. 1994). "Learning Long-Term Dependencies with Gradient Descent is Difficult". In: *Trans. Neur. Netw.* 5.2, pp. 157–166. ISSN: 1045-9227. DOI: 10.1109/72.279181. URL: https://doi.org/10.1109/72.279181.

Bengio, Yoshua, Tristan Deleu, Nasim Rahaman, Rosemary Ke, Sébastien Lachapelle, Olexa Bilaniuk, Anirudh Goyal, and Christopher Pal (2019). "A meta-transfer objective for learning to disentangle causal mechanisms". In: *arXiv preprint arXiv:1901.10912*.

Berner, Christopher, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. (2019). "Dota 2 with large scale deep reinforcement learning". In: *arXiv preprint arXiv:1912.06680*.

Berseth, Glen, Daniel Geng, Coline Manon Devin, Nicholas Rhinehart, Chelsea Finn, Dinesh Jayaraman, and Sergey Levine (2021). "SMiRL: Surprise Minimizing Reinforcement Learning in Unstable Environments". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=cPZOyoDloxl.

Bogacz, Rafal (2017). "A tutorial on the free-energy framework for modelling perception and learning". In: *Journal of Mathematical Psychology* 76. Model-based Cognitive Neuroscience,

pp. 198–211. ISSN: 0022-2496. DOI: https://doi.org/10.1016/j.jmp.2015.11.003. URL: https://www.sciencedirect.com/science/article/pii/S0022249615000759.

Bornschein, Jörg, Andriy Mnih, Daniel Zoran, and Danilo J Rezende (2017). "Variational memory addressing in generative models". In: *arXiv preprint arXiv:1709.07116*.

Botvinick, Matthew and James An (2009). "Goal-directed decision making in prefrontal cortex: A computational framework". In: *Advances in neural information processing systems* 21, pp. 169–176. ISSN: 1049-5258.

Brown, Tom et al. (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Ed. by H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin. Vol. 33. Curran Associates, Inc., pp. 1877–1901. URL: https://proceedings.neurips.cc/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Buckley, Christopher L., Chang Sub Kim, Simon McGregor, and Anil K. Seth (2017). "The free energy principle for action and perception: A mathematical review". In: *Journal of Mathematical Psychology* 81, pp. 55–79. ISSN: 0022-2496. DOI: https://doi.org/10.1016/j.jmp.2017.09.004. URL: http://www.sciencedirect.com/science/article/pii/S0022249617300962.

Butz, Martin V, David Bilkey, Dania Humaidan, Alistair Knott, and Sebastian Otte (2019). "Learning, planning, and control in a monolithic neural event inference architecture". In: *Neural Networks* 117, pp. 135–144.

Çatal, Ozan, Tim Verbelen, Johannes Nauta, Cedric De Boom, and Bart Dhoedt (2020). "Learning perception and planning with deep active inference". In: *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, pp. 3952–3956.

Çatal, Ozan, Tim Verbelen, Toon Van de Maele, Bart Dhoedt, and Adam Safron (2021). "Robot navigation as hierarchical active inference". In: *Neural Networks* 142, pp. 192–204. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2021.05.010. URL: https://www.sciencedirect.com/science/article/pii/S0893608021002021.

Chang, Bo, Minmin Chen, Eldad Haber, and Ed H. Chi (2019). *AntisymmetricRNN: A Dynamical System View on Recurrent Neural Networks*. arXiv: 1902.09689 [stat.ML].

Cho, Kyunghyun, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio (2014). "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078. arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

Chung, Junyoung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio (2015). "A Recurrent Latent Variable Model for Sequential Data". In: *Advances in Neural Information Processing Systems*. Ed. by C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett. Vol. 28. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2015/file/b618c3210e934362ac261db280128c22-Paper.pdf.

Ciria, Alejandra, G. Schillaci, G. Pezzulo, V. Hafner, and B. Lara (2021). "Predictive Processing in Cognitive Robotics: a Review". In: *Neural Comput.* 33, pp. 1402–1432.

Clark, Andy (2013). "Whatever next? Predictive brains, situated agents, and the future of cognitive science". In: *Behavioral and Brain Sciences* 36.3, pp. 181–204. DOI: 10.1017/S0140525X12000477.

Collins, Jasmine, Jascha Sohl-Dickstein, and David Sussillo (2017). *Capacity and Trainability in Recurrent Neural Networks*. arXiv: 1611.09913 [stat.ML].

Collobert, Ronan and Jason Weston (2008). "A unified architecture for natural language processing: Deep neural networks with multitask learning". In: *Proceedings of the 25th international conference on Machine learning*, pp. 160–167.

Creem-Regehr, Sarah H. (2009). "Sensory-motor and cognitive functions of the human posterior parietal cortex involved in manual actions". In: *Neurobiology of Learning and Memory* 91.2. Special Issue: Parietal Cortex, pp. 166–171. ISSN: 1074-7427. DOI: https://doi.org/10.1016/j.nlm.2008.10.004. URL: http://www.sciencedirect.com/science/article/pii/S1074742708001883.

Crick, Francis (Jan. 1989). "The recent excitement about neural networks". In: *Nature* 337.6203, pp. 129–132. ISSN: 1476-4687. DOI: 10.1038/337129a0. URL: https://doi.org/10.1038/337129a0.

Daucé, Emmanuel (2020). "End-Effect Exploration Drive for Effective Motor Learning". In: *International Workshop on Active Inference*. Springer, pp. 114–124.

Dayan, Peter and Geoffrey E. Hinton (1996). "Varieties of Helmholtz Machine". In: *Neural Networks* 9.8. Four Major Hypotheses in Neuroscience, pp. 1385–1403. ISSN: 0893-6080. DOI: https://doi.org/10.1016/S0893-6080(96)00009-3. URL: https://www.sciencedirect.com/science/article/pii/S0893608096000093.

Dayan, Peter, Geoffrey E. Hinton, Radford M. Neal, and Richard S. Zemel (Sept. 1995). "The Helmholtz Machine". In: *Neural Computation* 7.5, pp. 889–904. ISSN: 0899-7667. DOI: 10.1162/neco.1995.7.5.889. eprint: https://direct.mit.edu/neco/article-pdf/7/5/889/813131/neco.1995.7.5.889.pdf. URL: https://doi.org/10.1162/neco.1995.7.5.889.

Demircigil, Mete, Judith Heusel, Matthias Löwe, Sven Upgang, and Franck Vermet (2017). "On a model of associative memory with huge storage capacity". In: *Journal of Statistical Physics* 168.2, pp. 288–299.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2018). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *CoRR* abs/1810.04805. arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

Dilokthanakul, Nat, Pedro AM Mediano, Marta Garnelo, Matthew CH Lee, Hugh Salimbeni, Kai Arulkumaran, and Murray Shanahan (2016). "Deep unsupervised clustering with gaussian mixture variational autoencoders". In: *arXiv preprint arXiv:1611.02648*.

Doersch, Carl (2021). *Tutorial on Variational Autoencoders*. arXiv: 1606.05908 [stat.ML].

Doya, K. (1993). "Bifurcations of Recurrent Neural Networks in Gradient Descent Learning". In: *IEEE Transactions on Neural Networks*.

Dua, D. and C. Graff (2019). "UCI Machine Learning Repository". In: *[http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science*.

Dudai, Y. (2004). "The neurobiology of consolidations, or, how stable is the engram?" In: *Annual review of psychology* 55, pp. 51–86.

Elman, Jeffrey L. (1990). "Finding structure in time". In: *Cognitive Science* 14.2, pp. 179–211. ISSN: 0364-0213. DOI: https://doi.org/10.1016/0364-0213(90)90002-E. URL: https://www.sciencedirect.com/science/article/pii/036402139090002E.

Fabius, Otto and Joost R. van Amersfoort (2015). *Variational Recurrent Auto-Encoders*. arXiv: 1412.6581 [stat.ML].

Feldman, Harriet and Karl Friston (2010). "Attention, uncertainty, and free-energy". In: *Frontiers in human neuroscience* 4, p. 215.

Fenske, Mark J., Elissa Aminoff, Nurit Gronau, and Moshe Bar (2006). "Chapter 1 Top-down facilitation of visual object recognition: object-based and context-based contributions". In: *Visual Perception*. Ed. by S. Martinez-Conde, S.L. Macknik, L.M. Martinez, J.-M. Alonso, and P.U. Tse. Vol. 155. Progress in Brain Research. Elsevier, pp. 3–21. DOI: https://doi.org/10.1016/S0079-6123(06)55001-0. URL: https://www.sciencedirect.com/science/article/pii/S0079612306550010.

Fraccaro, Marco, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther (2016). "Sequential Neural Models with Stochastic Layers". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2016/file/208e43f0e45c4c78cafadb83d2888cb6-Paper.pdf.

Friston, K.J., J. Daunizeau, and S.J. Kiebel (2009). "Reinforcement Learning or Active Inference?" In: *PLoS ONE* 4.7, e6421.

Friston, K.J., T. FitzGerald, F. Rigoli, P. Schwartenbeck, J. O'Doherty, and G. Pezzulo (2016). "Active inference and learning". In: *Neuroscience & Biobehavioral Reviews* 68, pp. 862–879.

Friston, K.J. and S. Kiebel (2009). "Predictive coding under the free-energy principle". In: *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences* 364, pp. 1211–21.

Friston, K.J. and J. Kilner (2006). "A free energy principle for the brain". In: *J. Physiol. Paris* 100, pp. 70–87.

Friston, K.J., N. Trujillo-Barreto, and J. Daunizeau (2008). "DEM: A variational treatment of dynamic systems". In: *NeuroImage* 41.3, pp. 849–885. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2008.02.054. URL: https://www.sciencedirect.com/science/article/pii/S1053811908001894.

Friston, Karl (2003). "Learning and inference in the brain". In: *Neural Networks* 16.9. Neuroinformatics, pp. 1325–1352. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2003.06.005. URL: https://www.sciencedirect.com/science/article/pii/S0893608003002454.

— (2005). "A theory of cortical responses". In: *Philosophical Transactions of the Royal Society B: Biological Sciences* 360.1456, pp. 815–836. DOI: 10.1098/rstb.2005.1622. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rstb.2005.1622. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rstb.2005.1622.

Friston, Karl (Nov. 2008a). "Hierarchical Models in the Brain". In: *PLOS Computational Biology* 4.11, pp. 1–24.
— (2009). "The free-energy principle: a rough guide to the brain?" In: *Trends in Cognitive Sciences* 13.7, pp. 293–301. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2009.04.005. URL: https://www.sciencedirect.com/science/article/pii/S136466130900117X.
— (Aug. 2010a). "Is the free-energy principle neurocentric?" In: *Nature Reviews Neuroscience* 11.8, pp. 605–605. ISSN: 1471-0048. DOI: 10.1038/nrn2787-c2. URL: https://doi.org/10.1038/nrn2787-c2.
— (Feb. 2010b). "The free-energy principle: a unified brain theory?" In: *Nature Reviews Neuroscience* 11.2, pp. 127–138. ISSN: 1471-0048. DOI: 10.1038/nrn2787. URL: https://doi.org/10.1038/nrn2787.
Friston, Karl, Lancelot Da Costa, Danijar Hafner, Casper Hesp, and Thomas Parr (2021). "Sophisticated inference". In: *Neural Computation* 33.3, pp. 713–763.
Friston, Karl, Jérémie Mattout, and James Kilner (2011). "Action understanding and active inference". In: *Biological cybernetics* 104.1, pp. 137–160.
Friston, Karl, Jérémie Mattout, Nelson Trujillo-Barreto, John Ashburner, and Will Penny (2007). "Variational free energy and the Laplace approximation". In: *NeuroImage* 34.1, pp. 220–234. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2006.08.035. URL: https://www.sciencedirect.com/science/article/pii/S1053811906008822.
Friston, Karl, Francesco Rigoli, Dimitri Ognibene, Christoph Mathys, Thomas Fitzgerald, and Giovanni Pezzulo (2015). "Active inference and epistemic value". In: *Cognitive neuroscience* 6.4, pp. 187–214.
Friston, Karl, Christopher Thornton, and Andy Clark (2012). "Free-energy minimization and the dark-room problem". In: *Frontiers in psychology* 3, p. 130.
Friston, Karl J (2008b). "Variational filtering". In: *NeuroImage* 41.3, pp. 747–766.
Friston, Karl J. and K. Stephan (2007). "Free-energy and the brain". In: *Synthese* 159, pp. 417–458.
Fuster, J M (1973). "Unit activity in prefrontal cortex during delayed-response performance: neuronal correlates of transient memory." In: *Journal of Neurophysiology* 36.1. PMID: 4196203, pp. 61–78. DOI: 10.1152/jn.1973.36.1.61. eprint: https://doi.org/10.1152/jn.1973.36.1.61. URL: https://doi.org/10.1152/jn.1973.36.1.61.
Gemici, Mevlana, Chia-Chun Hung, Adam Santoro, Greg Wayne, Shakir Mohamed, Danilo J Rezende, David Amos, and Timothy Lillicrap (2017). "Generative temporal models with memory". In: *arXiv preprint arXiv:1702.04649*.
Girin, Laurent, Simon Leglaive, Xiaoyu Bie, Julien Diard, Thomas Hueber, and Xavier Alameda-Pineda (2020). "Dynamical variational autoencoders: A comprehensive review". In: *arXiv preprint arXiv:2008.12595*.
Goyal, Anirudh, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio (2017). "Z-Forcing: Training Stochastic Recurrent Networks". In: *Advances in Neural Information Processing Systems*. Ed. by I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett. Vol. 30. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2017/file/900c563bfd2c48c16701acca83ad858a-Paper.pdf.
Graves, Alex, Greg Wayne, and Ivo Danihelka (2014). "Neural turing machines". In: *arXiv preprint arXiv:1410.5401*.
Graves, Alex, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. (2016). "Hybrid computing using a neural network with dynamic external memory". In: *Nature* 538.7626, pp. 471–476.
Greff, Klaus, Rupesh Kumar Srivastava, Jan Koutnik, Bas R. Steunebrink, and Jürgen Schmidhuber (2015). "LSTM: A Search Space Odyssey". In: *CoRR* abs/1503.04069. arXiv: 1503.04069. URL: http://arxiv.org/abs/1503.04069.
Gregor, Karol, Ivo Danihelka, Alex Graves, and Daan Wierstra (2015). "DRAW: A Recurrent Neural Network For Image Generation". In: *CoRR* abs/1502.04623. arXiv: 1502.04623. URL: http://arxiv.org/abs/1502.04623.
He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2016). "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778.

He, Xu and Herbert Jaeger (2018). "Overcoming Catastrophic Interference using Conceptor-Aided Backpropagation". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=B1al7jg0b.

Hochreiter, Sepp and Jürgen Schmidhuber (Nov. 1997). "Long Short-Term Memory". In: *Neural Comput.* 9.8, pp. 1735–1780. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.8.1735. URL: https://doi.org/10.1162/neco.1997.9.8.1735.

Hoerzer, Gregor M., Robert Legenstein, and Wolfgang Maass (Nov. 2012). "Emergence of Complex Computational Structures From Chaotic Neural Networks Through Reward-Modulated Hebbian Learning". In: *Cerebral Cortex* 24.3, pp. 677–690. ISSN: 1047-3211. DOI: 10.1093/cercor/bhs348. eprint: https://academic.oup.com/cercor/article-pdf/24/3/677/14099466/bhs348.pdf. URL: https://doi.org/10.1093/cercor/bhs348.

Hopfield, J J (1982). "Neural networks and physical systems with emergent collective computational abilities". In: *Proceedings of the National Academy of Sciences* 79.8, pp. 2554–2558. ISSN: 0027-8424. DOI: 10.1073/pnas.79.8.2554. eprint: https://www.pnas.org/content/79/8/2554.full.pdf. URL: https://www.pnas.org/content/79/8/2554.

Hosoya, Toshihiko, Stephen A. Baccus, and Markus Meister (July 2005). "Dynamic predictive coding by the retina". In: *Nature* 436.7047, pp. 71–77. DOI: 10.1038/nature03689.

Houthooft, Rein, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel (2016). "Vime: Variational information maximizing exploration". In: *arXiv preprint arXiv:1605.09674*.

Hug, Ronny, Stefan Becker, Wolfgang Hübner, and Michael Arens (2021). *Quantifying the Complexity of Standard Benchmarking Datasets for Long-Term Human Trajectory Prediction*. arXiv: 2005.13934 [cs.CV].

Hughes, H.C., T.M. Darcey, H.I. Barkan, P.D. Williamson, D.W. Roberts, and C.H. Aslin (2001). "Responses of Human Auditory Association Cortex to the Omission of an Expected Acoustic Event". In: *NeuroImage* 13.6, pp. 1073–1089. ISSN: 1053-8119. DOI: https://doi.org/10.1006/nimg.2001.0766. URL: https://www.sciencedirect.com/science/article/pii/S1053811901907669.

Hwang, Jungsik, Jinhyung Kim, Ahmadreza Ahmadi, Minkyu Choi, and Jun Tani (2020). "Dealing With Large-Scale Spatio-Temporal Patterns in Imitative Interaction Between a Robot and a Human by Using the Predictive Coding Framework". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 50.5, pp. 1918–1931. DOI: 10.1109/TSMC.2018.2791984.

Ikeda, Kensuke, Kenju Otsuka, and Kenji Matsumoto (June 1989). "Maxwell-Bloch Turbulence". In: *Progress of Theoretical Physics Supplement* 99, pp. 295–324. ISSN: 0375-9687. DOI: 10.1143/PTPS.99.295.

Inoue, Katsuma, Kohei Nakajima, and Yasuo Kuniyoshi (2020). "Designing spontaneous behavioral switching via chaotic itinerancy". In: *Science Advances* 6.46. DOI: 10.1126/sciadv.abb3989. eprint: https://advances.sciencemag.org/content/6/46/eabb3989.full.pdf. URL: https://advances.sciencemag.org/content/6/46/eabb3989.

Ito, Masato and Jun Tani (2004). "On-line Imitative Interaction with a Humanoid Robot Using a Dynamic Neural Network Model of a Mirror System". In: *Adaptive Behavior* 12.2, pp. 93–115. DOI: 10.1177/105971230401200202. eprint: https://doi.org/10.1177/105971230401200202. URL: https://doi.org/10.1177/105971230401200202.

Jacquey, L., G. Baldassare, V.G. Santucci, and O'Reagan J.K. (2019). "Sensorimotor Contingencies as a Key Drive of Development: From Babies to Robots". In: *Frontiers in NeuroRobotics* 13.98. DOI: 10.3389/fnbot.2019.00098.

Jaeger, Herbert (Jan. 2001). "The "Echo State" Approach to Analysing and Training Recurrent Neural Networks". In: *GMD-Report 148, German National Research Institute for Computer Science*.

— (2014a). "Conceptors: an easy introduction". In: *CoRR* abs/1406.2671. arXiv: 1406.2671. URL: http://arxiv.org/abs/1406.2671.

— (2014b). "Controlling Recurrent Neural Networks by Conceptors". In: *CoRR* abs/1403.3369. arXiv: 1403.3369. URL: http://arxiv.org/abs/1403.3369.

Jehee, Janneke F M and Dana H Ballard (May 2009). "Predictive feedback can account for biphasic responses in the lateral geniculate nucleus". In: *PLoS computational biology* 5.5, e1000373. ISSN: 1553-734X. DOI: 10.1371/journal.pcbi.1000373. URL: https://europepmc.org/articles/PMC2670540.

Jung, Minju, Takazumi Matsumoto, and Jun Tani (2019). "Goal-Directed Behavior under Variational Predictive Coding: Dynamic Organization of Visual Attention and Working Memory". In: *CoRR* abs/1903.04932.

Kalchbrenner, Nal, Edward Grefenstette, and Phil Blunsom (2014). "A convolutional neural network for modelling sentences". In: *arXiv preprint arXiv:1404.2188*.

Kaneko, Kunihiko (1990). "Clustering, coding, switching, hierarchical ordering, and control in a network of chaotic elements". In: *Physica D: Nonlinear Phenomena* 41.2, pp. 137–172. ISSN: 0167-2789. DOI: 10.1016/0167-2789(90)90119-A.

Kaneko, Kunihiko and Ichiro Tsuda (2003). "Chaotic itinerancy". In: *Chaos: An Interdisciplinary Journal of Nonlinear Science* 13.3, pp. 926–936. DOI: 10.1063/1.1607783.

Kanerva, Pentti (1988). *Sparse distributed memory*. MIT press.

Ke, Nan Rosemary, Olexa Bilaniuk, Anirudh Goyal, Stefan Bauer, Hugo Larochelle, Bernhard Schölkopf, Michael C Mozer, Chris Pal, and Yoshua Bengio (2019). "Learning neural causal models from unknown interventions". In: *arXiv preprint arXiv:1910.01075*.

Kingma, Diederik P. and M. Welling (2014). "Auto-Encoding Variational Bayes". In: *CoRR* abs/1312.6114.

Kirkpatrick, James et al. (2017). "Overcoming catastrophic forgetting in neural networks". In: *Proceedings of the National Academy of Sciences* 114.13, pp. 3521–3526. ISSN: 0027-8424. DOI: 10.1073/pnas.1611835114. eprint: https://www.pnas.org/content/114/13/3521.full.pdf. URL: https://www.pnas.org/content/114/13/3521.

Klyubin, Alexander S, Daniel Polani, and Chrystopher L Nehaniv (2005). "All else being equal be empowered". In: *European Conference on Artificial Life*. Springer, pp. 744–753.

Knill, David C. and Alexandre Pouget (2004). "The Bayesian brain: the role of uncertainty in neural coding and computation". In: *Trends in Neurosciences* 27.12, pp. 712–719. ISSN: 0166-2236. DOI: https://doi.org/10.1016/j.tins.2004.10.007. URL: http://www.sciencedirect.com/science/article/pii/S0166223604003352.

Kohonen, T. (1982). "Self-organized formation of topologically correct feature maps". In: *Biological Cybernetics* 43, pp. 59–69.

Koutnik, Jan, Klaus Greff, Faustino J. Gomez, and Jurgen Schmidhuber (2014). "A Clockwork RNN". In: *CoRR* abs/1402.3511. arXiv: 1402.3511. URL: http://arxiv.org/abs/1402.3511.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*. NIPS'12. Lake Tahoe, Nevada: Curran Associates Inc., pp. 1097–1105.

Laje, R. and D.V. Buonomano (2013). "Robust timing and motor patterns by taming chaos in recurrent neural networks". In: *Nature Neuroscience* 16.7, pp. 925–935.

Lanillos, Pablo, Gordon Cheng, et al. (2020). "Robot self/other distinction: active inference meets neural networks learning in a mirror". In: *arXiv preprint arXiv:2004.05473*.

Launay, Julien, Iacopo Poli, Kilian Müller, Gustave Pariente, Igor Carron, Laurent Daudet, Florent Krzakala, and Sylvain Gigan (2020). "Hardware Beyond Backpropagation: a Photonic Co-Processor for Direct Feedback Alignment". In: *arXiv preprint arXiv:2012.06373*.

Le, Quoc V., Navdeep Jaitly, and Geoffrey E. Hinton (2015). "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units". In: *CoRR* abs/1504.00941. arXiv: 1504.00941. URL: http://arxiv.org/abs/1504.00941.

Lee, Dong-Hyun, Saizheng Zhang, Asja Fischer, and Yoshua Bengio (2015). "Difference Target Propagation". In: *Machine Learning and Knowledge Discovery in Databases*. Ed. by Annalisa Appice, Pedro Pereira Rodrigues, Vítor Santos Costa, Carlos Soares, João Gama, and Alípio Jorge. Cham: Springer International Publishing, pp. 498–515. ISBN: 978-3-319-23528-8.

Li, Zhizhong and Derek Hoiem (2017). "Learning without forgetting". In: *IEEE transactions on pattern analysis and machine intelligence* 40.12, pp. 2935–2947.

Lillicrap, Timothy P, Daniel Cownden, Douglas B Tweed, and Colin J Akerman (Nov. 2016). "Random synaptic feedback weights support error backpropagation for deep learning". In: *Nature communications* 7, p. 13276. ISSN: 2041-1723. DOI: 10.1038/ncomms13276. URL: https://europepmc.org/articles/PMC5105169.

Lotter, William, Gabriel Kreiman, and David D. Cox (2016). "Deep Predictive Coding Networks for Video Prediction and Unsupervised Learning". In: *CoRR* abs/1605.08104. arXiv: 1605.08104. URL: http://arxiv.org/abs/1605.08104.

Lukoševičius, Mantas and Herbert Jaeger (2009). "Reservoir computing approaches to recurrent neural network training". In: *Computer Science Review* 3.3, pp. 127–149. ISSN: 1574-0137. DOI: https://doi.org/10.1016/j.cosrev.2009.03.005.

Maass, W., T. Natschläger, and H. Markram (2002). "Real-Time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations". In: *Neural Computation* 14.11, pp. 2531–2560. DOI: 10.1162/089976602760407955.

Mallya, Arun, Dillon Davis, and Svetlana Lazebnik (2018). "Piggyback: Adapting a single network to multiple tasks by learning to mask weights". In: *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 67–82.

Mannella, Francesco and Gianluca Baldassarre (Dec. 2015). "Selection of Cortical Dynamics for Motor Behaviour by the Basal Ganglia". In: *Biol. Cybern.* 109.6, pp. 575–595. ISSN: 0340-1200. DOI: 10.1007/s00422-015-0662-6. URL: https://doi.org/10.1007/s00422-015-0662-6.

Martens, J. (2010). "Deep learning via Hessian-free optimization". In: *ICML*.

Martens, James and Ilya Sutskever (2011). "Learning recurrent neural networks with hessian-free optimization". In: *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 1033–1040.

Matsumoto, Takazumi and Jun Tani (2020). "Goal-directed planning for habituated agents by active inference using a variational recurrent neural network". In: *Entropy* 22.5, p. 564.

Meo, Cristian and Pablo Lanillos (2021). "Multimodal VAE Active Inference Controller". In: *arXiv preprint arXiv:2103.04412*.

Mikolov, Tomas, Armand Joulin, Sumit Chopra, Michael Mathieu, and Marc'Aurelio Ranzato (Dec. 2014). "Learning Longer Memory in Recurrent Neural Networks". In: *arXiv e-prints*, arXiv:1412.7753, arXiv:1412.7753. arXiv: 1412.7753 [cs.NE].

Millidge, Beren (2020). "Deep active inference as variational policy gradients". In: *Journal of Mathematical Psychology* 96, p. 102348.

Millidge, Beren, Alexander Tschantz, and Christopher L Buckley (2021). "Whence the expected free energy?" In: *Neural Computation* 33.2, pp. 447–482.

— (2020a). "Predictive Coding Approximates Backprop along Arbitrary Computation Graphs". In: *CoRR* abs/2006.04182. arXiv: 2006.04182. URL: https://arxiv.org/abs/2006.04182.

Millidge, Beren, Alexander Tschantz, Anil Seth, and Christopher L Buckley (2020b). *Relaxing the Constraints on Predictive Coding Models*. arXiv: 2010.01047 [q-bio.NC].

Mirowski, Piotr, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, et al. (2016). "Learning to navigate in complex environments". In: *arXiv preprint arXiv:1611.03673*.

Mishkin, Dmytro and Jiri Matas (2016). *All you need is a good init*. arXiv: 1511.06422 [cs.LG].

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller (2013). "Playing Atari with Deep Reinforcement Learning". In: *CoRR* abs/1312.5602. arXiv: 1312.5602. URL: http://arxiv.org/abs/1312.5602.

Mochizuki, K., S. Nishide, H. G. Okuno, and T. Ogata (2013). "Developmental Human-Robot Imitation Learning of Drawing with a Neuro Dynamical System". In: *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2336–2341. DOI: 10.1109/SMC.2013.399.

Müller, James R., Andrew B. Metha, John Krauskopf, and Peter Lennie (1999). "Rapid Adaptation in Visual Cortex to the Structure of Images". In: *Science* 285.5432, pp. 1405–1408. ISSN: 0036-8075. DOI: 10.1126/science.285.5432.1405. eprint: https://science.sciencemag.org/content/285/5432/1405.full.pdf. URL: https://science.sciencemag.org/content/285/5432/1405.

Mushiake, Hajime, Naohiro Saito, Kazuhiro Sakamoto, Yasuto Itoyama, and Jun Tanji (2006). "Activity in the Lateral Prefrontal Cortex Reflects Multiple Steps of Future Events in Action Plans". In: *Neuron* 50.4, pp. 631–641. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2006.03.045. URL: http://www.sciencedirect.com/science/article/pii/S0896627306002728.

Nagabandi, Anusha, Kurt Konolige, Sergey Levine, and Vikash Kumar (2020). "Deep dynamics models for learning dexterous manipulation". In: *Conference on Robot Learning*. PMLR, pp. 1101–1112.

Namikawa, Jun, Ryunosuke Nishimoto, and Jun Tani (Oct. 2011). "A Neurodynamic Account of Spontaneous Behaviour". In: *PLOS Computational Biology* 7.10, pp. 1–13. DOI: 10.1371/journal.pcbi.1002221.

Nøkland, Arild (2016). "Direct Feedback Alignment Provides Learning in Deep Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., pp. 1037–1045. URL: https://proceedings.neurips.cc/paper/2016/file/d490d7b4576290fa60eb31b5fc917ad1-Paper.pdf.

O'Regan, J. Kevin and Alva Noë (2001). "A sensorimotor account of vision and visual consciousness". In: *Behavioral and Brain Sciences* 24.5, pp. 939–973. DOI: 10.1017/S0140525X01000115.

Ohata, Wataru and Jun Tani (2020). "Investigation of multimodal and agential interactions in human-robot imitation, based on frameworks of predictive coding and active inference". In: *arXiv preprint arXiv:2002.01632* 2.

Oliver, Guillermo, Pablo Lanillos, and Gordon Cheng (2019). "Active inference body perception and action for humanoid robots". In: *CoRR* abs/1906.03022.

Ororbia, A., A. Mali, C. L. Giles, and D. Kifer (2020). "Continual Learning of Recurrent Neural Networks by Locally Aligning Distributed Representations". In: *IEEE Transactions on Neural Networks and Learning Systems* 31.10, pp. 4267–4278.

Ororbia, Alexander and Daniel Kifer (2020). "The Neural Coding Framework for Learning Generative Models". In: *CoRR* abs/2012.03405. arXiv: 2012.03405. URL: https://arxiv.org/abs/2012.03405.

Otte, Sebastian, Theresa Schmitt, Karl Friston, and Martin V Butz (2017). "Inferring adaptive goal-directed behavior within recurrent neural networks". In: *International Conference on Artificial Neural Networks*. Springer, pp. 227–235.

Ouden, Hanneke E.M. den, Karl J. Friston, Nathaniel D. Daw, Anthony R. McIntosh, and Klaas E. Stephan (Sept. 2008). "A Dual Role for Prediction Error in Associative Learning". In: *Cerebral Cortex* 19.5, pp. 1175–1185. ISSN: 1047-3211. DOI: 10.1093/cercor/bhn161. eprint: https://academic.oup.com/cercor/article-pdf/19/5/1175/17301930/bhn161.pdf. URL: https://doi.org/10.1093/cercor/bhn161.

Oudeyer, Pierre-Yves and Frederic Kaplan (2009). "What is intrinsic motivation? A typology of computational approaches". In: *Frontiers in neurorobotics* 1, p. 6.

Pascanu, Razvan, Tomas Mikolov, and Yoshua Bengio (2012). "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063.

— (2013). "On the Difficulty of Training Recurrent Neural Networks". In: *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*. ICML'13. Atlanta, GA, USA: JMLR.org, III–1310–III–1318.

Perez, Christian, Felipe Petroski Such, and Theofanis Karaletsos (2020). "Generalized hidden parameter mdps: Transferable model-based rl in a handful of trials". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04, pp. 5403–5411.

Pezzato, Corrado, Riccardo Ferrari, and Carlos Hernández Corbato (2020). "A novel adaptive controller for robot manipulators based on active inference". In: *IEEE Robotics and Automation Letters* 5.2, pp. 2973–2980.

Pezzulo, Giovanni and Paul Cisek (2016). "Navigating the Affordance Landscape: Feedback Control as a Process Model of Behavior and Cognition". In: *Trends in Cognitive Sciences* 20.6, pp. 414–424. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2016.03.013. URL: http://www.sciencedirect.com/science/article/pii/S1364661316300067.

Pio-Lopez, Léo, Ange Nizard, Karl Friston, and Giovanni Pezzulo (2016). "Active inference and robot control: a case study". In: *Journal of The Royal Society Interface* 13.122, p. 20160616. DOI: 10.1098/rsif.2016.0616. eprint: https://royalsocietypublishing.org/doi/pdf/10.1098/rsif.2016.0616. URL: https://royalsocietypublishing.org/doi/abs/10.1098/rsif.2016.0616.

Pitti, Alexandre, Philippe Gaussier, and Mathias Quoy (Mar. 2017). "Iterative free-energy optimization for recurrent neural networks (INFERNO)". In: *PLOS ONE* 12.3, pp. 1–33. DOI: 10.1371/journal.pone.0173684. URL: https://doi.org/10.1371/journal.pone.0173684.

Planton, Samuel, Marieke Longcamp, Patrice Péran, Jean-François Démonet, and Mélanie Jucla (2017). "How specialized are writing-specific brain regions? An fMRI study of writing, drawing and oral spelling". In: *Cortex* 88, pp. 66–80. ISSN: 0010-9452. DOI: https://doi.org/10.1016/j.cortex.2016.11.018. URL: http://www.sciencedirect.com/science/article/pii/S0010945216303458.

Pyle, Ryan and Robert Rosenbaum (2019). "A reservoir computing model of reward-modulated motor learning and automaticity". In: *Neural computation* 31.7, pp. 1430–1461.

Ramsauer, Hubert, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Thomas Adler, Lukas Gruber, Markus Holzleitner, Milena Pavlović, Geir Kjetil Sandve, et al. (2020). "Hopfield networks is all you need". In: *arXiv preprint arXiv:2008.02217*.

Rao, R.P and D. Ballard (1999). "Predictive coding in the visual cortex a functional interpretation of some extra-classical receptive-field effects". In: *Nat Neurosci* 2, pp. 79–87.

Rao, Rajesh P. N. and Dana H. Ballard (May 1997). "Dynamic Model of Visual Recognition Predicts Neural Response Properties in the Visual Cortex". In: *Neural Computation* 9.4, pp. 721–763. ISSN: 0899-7667. DOI: 10.1162/neco.1997.9.4.721. eprint: https://direct.mit.edu/neco/article-pdf/9/4/721/1062248/neco.1997.9.4.721.pdf. URL: https://doi.org/10.1162/neco.1997.9.4.721.

Rebuffi, Sylvestre-Alvise, Alexander Kolesnikov, Georg Sperl, and Christoph H Lampert (2017). "icarl: Incremental classifier and representation learning". In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010.

Rezende, Danilo Jimenez, Shakir Mohamed, and Daan Wierstra (2014). *Stochastic Backpropagation and Approximate Inference in Deep Generative Models*. arXiv: 1401.4082 [stat.ML].

Rinne, Teemu, Alexander Degerman, and Kimmo Alho (2005). "Superior temporal and inferior frontal cortices are activated by infrequent sound duration decrements: an fMRI study". In: *NeuroImage* 26.1, pp. 66–72. ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2005.01.017. URL: https://www.sciencedirect.com/science/article/pii/S1053811905000479.

Rumelhart, D.E., G.E. Hinton, and R.J. Williams (1986). "Learning representations by back-propagating errors". In: *Nature* 323.6088, pp. 533–536.

Sancaktar, Cansu, Marcel AJ van Gerven, and Pablo Lanillos (2020). "End-to-end pixel-based deep active inference for body perception and action". In: *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, pp. 1–8.

Schillaci, Guido, Alejandra Ciria, and Bruno Lara (2020). "Tracking emotions: intrinsic motivation grounded on multi-level prediction error dynamics". In: *2020 Joint IEEE 10th International Conference on Development and Learning and Epigenetic Robotics (ICDL-EpiRob)*. IEEE, pp. 1–8.

Schmidhuber, Jürgen, Daan Wierstra, Matteo Gagliolo, and Faustino Gomez (2007). "Training Recurrent Networks by Evolino". In: *Neural Computation* 19.3, pp. 757–779. DOI: 10.1162/neco.2007.19.3.757.

Schmidhuber, Jürgen, Daan Wierstra, and Faustino Gomez (2005). "Evolino: Hybrid Neuroevolution / Optimal Linear Search for Sequence Learning". In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence*. IJCAI'05. Edinburgh, Scotland: Morgan Kaufmann Publishers Inc., pp. 853–858.

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, Karen Simonyan, Laurent Sifre, Simon Schmitt, Arthur Guez, Edward Lockhart, Demis Hassabis, Thore Graepel, et al. (2020). "Mastering atari, go, chess and shogi by planning with a learned model". In: *Nature* 588.7839, pp. 604–609.

Schröger, Erich, Anna Marzecová, and Iria SanMiguel (2015). "Attention and prediction in human audition: a lesson from cognitive psychophysiology". In: *European Journal of Neuroscience* 41.5, pp. 641–664. DOI: https://doi.org/10.1111/ejn.12816. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/ejn.12816. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/ejn.12816.

Shabanian, Samira, Devansh Arpit, Adam Trischler, and Yoshua Bengio (2017). *Variational Bi-LSTMs*. arXiv: 1711.05717 [stat.ML].

Shadmehr, Reza, Maurice A. Smith, and John W. Krakauer (2010). "Error Correction, Sensory Prediction, and Adaptation in Motor Control". In: *Annual Review of Neuroscience* 33.1. PMID: 20367317, pp. 89–108. DOI: 10.1146/annurev-neuro-060909-153135. eprint: https://doi.org/10.1146/annurev-neuro-060909-153135. URL: https://doi.org/10.1146/annurev-neuro-060909-153135.

Shin, Hanul, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim (2017). "Continual learning with deep generative replay". In: *arXiv preprint arXiv:1705.08690*.

Silver, David et al. (2016). "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529, pp. 484–503. URL: http://www.nature.com/nature/journal/v529/n7587/full/nature16961.html.

Silver, David et al. (2017). "Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm". In: *CoRR* abs/1712.01815. arXiv: 1712.01815. URL: http://arxiv.org/abs/1712.01815.

Smout, Cooper A., Matthew F. Tang, Marta I. Garrido, and Jason B. Mattingley (Feb. 2019). "Attention promotes the neural encoding of prediction errors". In: *PLOS Biology* 17.2, pp. 1–22. DOI: 10.1371/journal.pbio.2006812. URL: https://doi.org/10.1371/journal.pbio.2006812.

Sontakke, Sumedh A, Arash Mehrjou, Laurent Itti, and Bernhard Schölkopf (July 2021). "Causal Curiosity: RL Agents Discovering Self-supervised Experiments for Causal Representation Learning". In: *Proceedings of the 38th International Conference on Machine Learning*. Ed. by Marina Meila and Tong Zhang. Vol. 139. Proceedings of Machine Learning Research. PMLR, pp. 9848–9858. URL: https://proceedings.mlr.press/v139/sontakke21a.html.

Spratling, M. W., K. De Meyer, and R. Kompass (Jan. 2009). "Unsupervised Learning of Overlapping Image Components Using Divisive Input Modulation". In: *Intell. Neuroscience* 2009. ISSN: 1687-5265. DOI: 10.1155/2009/381457. URL: https://doi.org/10.1155/2009/381457.

Spratling, Michael (2008). "Reconciling predictive coding and biased competition models of cortical function". In: *Frontiers in Computational Neuroscience* 2, p. 4. ISSN: 1662-5188. DOI: 10.3389/neuro.10.004.2008. URL: https://www.frontiersin.org/article/10.3389/neuro.10.004.2008.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber (2015). "Highway Networks". In: *CoRR* abs/1505.00387. arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

Stefanics, Gábor, Jan Kremláček, and István Czigler (2014). "Visual mismatch negativity: a predictive coding view". In: *Frontiers in Human Neuroscience* 8, p. 666. ISSN: 1662-5161. DOI: 10.3389/fnhum.2014.00666. URL: https://www.frontiersin.org/article/10.3389/fnhum.2014.00666.

Summerfield, Christopher and Etienne Koechlin (2008). "A Neural Representation of Prior Information during Perceptual Inference". In: *Neuron* 59.2, pp. 336–347. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2008.05.021. URL: https://www.sciencedirect.com/science/article/pii/S089662730800456X.

Summerfield, Christopher, Emily H. Trittschuh, Jim M. Monti, M-Marsel Mesulam, and Tobias Egner (July 2008). "Neural repetition suppression reflects fulfilled perceptual expectations". In: *Nature Neuroscience* 11.9, pp. 1004–1006. ISSN: 1546-1726. DOI: 10.1038/nn.2163. URL: https://doi.org/10.1038/nn.2163.

Sun, Yi, Faustino Gomez, and Jürgen Schmidhuber (2011). "Planning to be surprised: Optimal bayesian exploration in dynamic environments". In: *International Conference on Artificial General Intelligence*. Springer, pp. 41–51.

Sun, Zekun and Chaz Firestone (2020). "The Dark Room Problem". In: *Trends in Cognitive Sciences* 24.5, pp. 346–348. ISSN: 1364-6613. DOI: https://doi.org/10.1016/j.tics.2020.02.006. URL: https://www.sciencedirect.com/science/article/pii/S1364661320300589.

Sussillo, David and L.F. Abbott (2009). "Generating Coherent Patterns of Activity from Chaotic Neural Networks". In: *Neuron* 63.4, pp. 544–557. ISSN: 0896-6273. DOI: https://doi.org/10.1016/j.neuron.2009.07.018. URL: https://www.sciencedirect.com/science/article/pii/S0896627309005479.

Sutskever, Ilya, James Martens, and Geoffrey Hinton (2011). "Generating Text with Recurrent Neural Networks". In: *Proceedings of the 28th International Conference on International Conference on Machine Learning*. ICML'11. Bellevue, Washington, USA: Omnipress, pp. 1017–1024. ISBN: 9781450306195.

Sutskever, Ilya, Oriol Vinyals, and Quoc V Le (2014). "Sequence to sequence learning with neural networks". In: *Advances in neural information processing systems*, pp. 3104–3112.

Tallec, Corentin and Yann Ollivier (2018). "Can recurrent neural networks warp time?" In: *CoRR* abs/1804.11188. arXiv: 1804.11188. URL: http://arxiv.org/abs/1804.11188.

Tani, J. and M. Ito (2003). "Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 33.4, pp. 481–488. DOI: 10.1109/TSMCA.2003.809171.

Taylor, Graham W. and Geoffrey E. Hinton (2009). "Factored Conditional Restricted Boltzmann Machines for Modeling Motion Style". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, pp. 1025–1032. ISBN: 9781605585161.

Tschantz, Alexander, Manuel Baltieri, Anil K Seth, and Christopher L Buckley (2020a). "Scaling active inference". In: *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, pp. 1–8.

Tschantz, Alexander, Anil K Seth, and Christopher L Buckley (2020b). "Learning action-oriented models through active inference". In: *PLoS computational biology* 16.4, e1007805.

Tsuda, Ichiro (1991). "Chaotic itinerancy as a dynamical basis of hermeneutics in brain and mind". In: *World Futures* 32.2-3, pp. 167–184. DOI: 10.1080/02604027.1991.9972257.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin (2017). "Attention is All You Need". In: URL: https://arxiv.org/pdf/1706.03762.pdf.

Verstraeten, D., B. Schrauwen, M. D'Haene, and D. Stroobandt (2007). "An experimental unification of reservoir computing methods". In: *Neural Network* 20, pp. 391–403.

Von Helmholtz, Hermann (1867). *Handbuch der physiologischen Optik: mit 213 in den Text eingedruckten Holzschnitten und 11 Tafeln*. Vol. 9. Voss.

Vorontsov, Eugene, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal (2017). "On orthogonality and learning recurrent networks with long term dependencies". In: *CoRR* abs/1702.00071. arXiv: 1702.00071. URL: http://arxiv.org/abs/1702.00071.

Wang, Wei, Sherwin S. Chan, Dustin A. Heldman, and Daniel W. Moran (2010). "Motor Cortical Representation of Hand Translation and Rotation during Reaching". In: *Journal of Neuroscience* 30.3, pp. 958–962. ISSN: 0270-6474. DOI: 10.1523/JNEUROSCI.3742-09.2010. eprint: https://www.jneurosci.org/content/30/3/958.full.pdf. URL: https://www.jneurosci.org/content/30/3/958.

Werbos, Paul J. (1982). "Applications of advances in nonlinear sensitivity analysis". In: *System Modeling and Optimization*. Ed. by R. F. Drenick and F. Kozin. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 762–770. ISBN: 978-3-540-39459-4.

— (1988). "Generalization of backpropagation with application to a recurrent gas market model". In: *Neural Networks* 1.4, pp. 339–356. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(88)90007-X. URL: https://www.sciencedirect.com/science/article/pii/089360808890007X.

Whittington, James C. R. and Rafal Bogacz (May 2017). "An Approximation of the Error Backpropagation Algorithm in a Predictive Coding Network with Local Hebbian Synaptic Plasticity". In: *Neural Computation* 29.5, pp. 1229–1262. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00949. eprint: https://direct.mit.edu/neco/article-pdf/29/5/1229/996846/neco_a_00949.pdf. URL: https://doi.org/10.1162/NECO%5C_a%5C_00949.

Wu, Yan, Greg Wayne, Alex Graves, and Timothy Lillicrap (2018). "The kanerva machine: A generative distributed memory". In: *arXiv preprint arXiv:1804.01756*.

Yamashita, Yuichi and Jun Tani (Nov. 2008). "Emergence of Functional Hierarchy in a Multiple Timescale Neural Network Model: A Humanoid Robot Experiment". In: *PLOS Computational Biology* 4.11, pp. 1–18. DOI: 10.1371/journal.pcbi.1000220. URL: https://doi.org/10.1371/journal.pcbi.1000220.

Yao, Kaisheng, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer (2015). "Depth-Gated LSTM". In: *CoRR* abs/1508.03790. arXiv: 1508.03790. URL: http://arxiv.org/abs/1508.03790.

Yue, Boxuan, Junwei Fu, and Jun Liang (2018). "Residual Recurrent Neural Networks for Learning Sequential Representations". In: *Information* 9.3. ISSN: 2078-2489. DOI: 10.3390/info9030056. URL: https://www.mdpi.com/2078-2489/9/3/56.

Zhu, Yuke, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi (2017). "Target-driven visual navigation in indoor scenes using deep reinforcement learning". In: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, pp. 3357–3364.