Deep learning models and algorithms for sequential data problems : applications to language modelling and uncertainty quantification

Alice Martin

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2022IPPAS007

Thèse de doctorat

Deep Learning models and algorithms for sequential data problems - Applications to Language Modelling and Uncertainty Quantification

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°574 École doctorale de mathématiques Hadamard (EDMH)
Spécialité de doctorat : Mathématiques Appliquées

Thèse présentée et soutenue à Saclay, le 14 Juin 2022, par

ALICE MARTIN

Composition du Jury :

François Desbouvries Professeur, Télécom Sud-Paris (SAMOVAR)	Président
Pierre Pudlo Professeur, I2M — Institut de Mathématiques de Marseille, UMR CNRS 7373	Rapporteur
François Septier Professeur, Université Bretagne Sud, LMBA UMR CNRS 6205	Rapporteur
Laure Soulier Maitresse de Conférences, Université de la Sorbonne, ISIR, UMR 7222	Examinatrice
Sylvain Le Corff Professeur, Télécom Sud-Paris (SAMOVAR)	Directeur de thèse
Olivier Pietquin Directeur de Recherche, Google Brain	Co-directeur de thèse
Charles Ollion Chercheur, Ecole Polytechnique (CMAP)	Invité
Florian Strub Chercheur, Deep Mind	Invité

Remerciements

Je remercie tant les personnes de mon entourage professionnel qui ont permis à cette thèse de se réaliser dans de bonnes conditions, et d'être poursuivie à son terme, que mon entourage qui m'a soutenue et entourée pendant ces trois ans.

Mes premiers remerciements vont à Eric Moulines, qui a rendu cette thèse possible en faisant un pari osé sur mon profil atypique, et a suivi de loin son déroulement tout du long pour faire en sorte qu'elle se passe bien. Je remercie ensuite mon directeur de thèse Sylvain Le Corff pour m'avoir accompagné et aidé de bout en bout, et notamment pour sa disponibilité, sa gentillesse, son indulgence et sa pédagogie. Je remercie dans la foulée Olivier Pietquin, mon co-directeur de thèse, qui a suggéré un sujet intéressant, a su donner des idées-clés pour améliorer ou initier des travaux de recherche, et donner de précieux conseils sur le déroulé général de la thèse. Un merci aussi à Charles Ollion et Florian Strub, avec qui j'ai eu l'occasion de collaborer pendant la majeure partie de cette thèse, et pour leurs précieux conseils tant au niveau du code, de la rédaction des papiers, ou de la gestion de vie d'un thésard. Je remercié également Pierre Gloaguen pour sa précieuse aide sur mes travaux sur le lissage particulaire, et bien sûr Guillaume Quispe, qui m'a épaulé sur le code, et sans qui la première publication de cette thèse n'aurait pas été possible. Enfin, je remercie les membres du jury, et en particulier les rapporteurs pour avoir lu et écouté mes travaux de recherche.

Je remercie bien sûr mes parents, Régis et Catherine, qui m'ont donné une éducation qui m'a permis d'atteindre ce niveau d'études, m'ont épaulé dans mon choix de revenir vers des "études" après trois ans dans l'industrie, m'ont accueilli de nombreuses nuits sous leur toit, se sont intéressés à ma thèse, et m'ont soutenu dans les moments de travail intense. Un merci aussi à mes trois sœurs, Camille, Clémence et Emilie, et ma grand-mère Odile, pour avoir patiemment écouté mes tribulations sur mon sujet de thèse, et le travail que cela engendrait, bien qu'elles soient très éloignées du domaine. Un grand merci et une chaude pensée pour mon copain François, qui a su surtout me faire décrocher du travail pour m'emmener partout en montagne, mais toujours en veillant que cela ne perturbe pas le déroulé de ma thèse. Je remercie mes amis de tous les horizons, pour m'avoir soutenu et surtout diverti pendant ces trois ans : Florine, Jessica, Sylvain B, Marie-Agnès, Marie-Astrid, Luis, Ollin, Nitish, Joe, Romain, Sharon, Bill, Paul, Janice, Thomas, Nono, Sylvain E, Christophe, Alban, Vincent B, Vincent L, Paulin, Elise, Nicolas, Kevin.

J'aimerais enfin dédier cette thèse à mon autre grand-mère, Hélène, qui a été d'un énorme support durant mon temps de télétravail dans les Alpes, et qui aujourd'hui est dans les derniers jours de sa vie.

Table des matières

List of Symbols

The next list describes the symbols for the main notations used within this thesis. The list is not exhaustive, it mentions recurrent notations that appear throughout multiple chapters and sections.

Symbols for Deep Learning Models - Sections 3.1 and 3.2.1

θ trainable set of parameters of a neural network

$J(\theta)$ cost function for training the neural network

h_t hidden state at time step t of a recurrent neural network

o_t logits output (i.e unnormalized logarithmic distribution) at time step t of a recurrent neural network

Q set of queries for the Transformer self-attention model

K set of keys for the Transformer self-attention model

V set of values for the Transformer self-attention model

Symbols for Language Models - Section 4.1 and Chapter 5

$w_{1:T} = (w_1, \dots, w_T)$ sequence of input words and more generally, for any sequence $\{a_u\}_{u \geq 1}$ and all $1 \leq s \leq t$, $a_{s:t} = (a_s, \dots, a_t)$

$p_\theta^{LM,(c)}$ conditional or unconditional generic Language Model

c context for a conditional Language Model

$\mathcal{V}$ discrete Vocabulary on which the probability distribution of a Language Model is defined

Symbols for Reinforcement Learning algorithms - Section 4.3 and Chapter 5

s_t action taken at time step t in a Markov Decision Process

a_t action taken at time step t in a Markov Decision Process

r_t reward received at timestep t by the reinforcement learning agent

π_θ policy of a reinforcement learning problem

A_t advantage function at timestep t of the Markov Decision Process

$\pi_{\theta_{old}}$ previous policy (before update) in the proximal policy optimization algorithm

Symbols for Sequential Monte Carlo algorithms - Section 3.3.1, Chapter 6 and Chapter 7

$(X_t)_{1\leq t\leq T}$ sequence of latent states of a state-space model

$(Y_t)_{1\leq t\leq T}$ sequence of noisy observations of a state-space model

$q_{\theta,t}$ transition density function for the state-space model at time t

$g_{\theta,t}$ conditional density of the observation of a state-space model at time t

$\ell_{\theta,t}$ product of the transition density and of the conditional density at time t

$\mathsf{L}_T(\theta)$ likelihood of the observations $(Y_t)_{1\leq t\leq T}$ in a state-space model

$\phi^{\theta}_{0:t|t'}$ conditional density of the latent states up to time step t given the observations until up to timestep t'

N number of particles in particle filtering algorithms

$\{\xi^i\}_{i=1}^N$ set of particle samples for particle filtering algorithms

$\{\omega^i\}_{i=1}^N$ set of resampling weights for particle filtering algorithms

I_t^i resampling index for particle ξ_t^i for particle filtering/smoothing algorithms

p_t^{θ} proposal distribution for sampling the next set of particles $\{\xi_t^i\}_{i=1}^N$ in particle filtering algorithm

$J_k^{(i,j)}$ backward sampling index in Chapter 7

$\tilde{N}$ number of backward samples in Chapter 7

Acronyms

AI Artificial Intelligence. 21, 68, 130

DGM Deep Generative Model. 12, 13, 22, 41

GAN Generative Adversarial Network. 12, 22, 41

HMM Hidden Markov Model. 41, 42, 107

KL Kullback Leibler. 42, 74

LM Language Model. 13, 23, 40, 51, 67, 130

LSTM Long-Term Short Memory. 35, 72, 91

MDP Markov Decision Process. 58

NLG Natural Language Generation. 22, 41, 57, 80, 85

NLP Natural Language Processing. 21, 36, 57, 73, 99, 130

NLU Natural Language Understanding. 21, 41, 57

PMS Poor man smoother. 46

PPO Proximal Policy Optimization. 61, 69

RL Reinforcement Learning. 14, 24, 57, 68, 129

RNN Recurrent Neural Network. 12, 22, 34, 98, 118

SL Supervised Learning. 13, 23, 56, 67

SMC Sequential Monte Carlo. 17, 27, 41, 55, 99, 108, 129

VAE Variational Auto-Encoder. 12, 22, 41

VQG Visual Question Generation. 69

Chapitre 1

Introduction (Version Française)

1.1 Contexte et Motivation

Le langage est une composante essentielle de nombreux systèmes d'Intelligence Artificielle (IA). Central à la pensée humaine, il permet d'encoder des abstractions, de généraliser concepts et objets, de communiquer des intentions et des besoins, à la fois à nous-mêmes et à des tierces personnes [108]. Développer des systèmes qui peuvent comprendre et produire du langage humain est l'un des principaux défis de la quête vers l'Intelligence Artificielle. C'est l'objectif du Traitement Automatique des Langues (TAL), un domaine à l'intersection entre la linguistique, l'informatique et l'Intelligence Artificielle. Aujourd'hui, la variété de systèmes de TAL reflète l'omniprésence du langage dans les systèmes d'aide à l'intelligence humaine. Les applications industrielles de TAL comprennent la complétion de texte, la recherche web, la traduction automatique, l'analyse de sentiments (à des fins marketing, pour détecter des comportements nocifs sur Internet, etc.), la publicité personnalisée, la lecture automatique de documents légaux et médicaux, les chatbots, etc.

Le Traitement Automatique des Langues. Au coeur de chaque tâche de TAL, réside l'important problème de la compréhension du langage [48]. Celui-ci comprend trois défis-clés : comprendre le processus de pensée, comprendre la représentation et le sens d'un élément linguistique, et assimiler la connaissance du monde. Dans le Traitement Automatique des Langues, on distingue par conséquent la notion de *syntaxe*, qui correspond à la grammaire et à la structure des phrases, la *sémantique* qui se concentre sur le sens des mots et des phrases, le *discours* qui s'intéresse à la structure de différents type de textes en s'appuyant sur leur structure, et le *pragmatisme* qui représente la connaissance du monde extérieur.

Depuis les années 1990, le domaine du TAL est dominé par l'approche Empirique, qui suppose que l'Homme apprend la structure détaillée du langage à travers l'expérience, et a donné naissance à des systèmes de TAL basés sur des méthodes d'apprentissage statistique sur de grands corpus de texte [1, 263, 86, 117]. Les années

2010 ont vu l'avènement des premiers systèmes de TAL statistiques performants et polyvalents, via l'émergence de l'apprentissage profond [294, 107] et le développement de réseaux de neurones spécialisés pour analyser du texte. Les architectures des Réseaux de Neurones Récurrents (RNN) [74, 125, 46], basées sur l'idée de représenter de la donnée séquentielle, peuvent facilement modéliser des textes de longueur variable, notamment des longues phrases, des paragraphes ou des documents entiers. Ces architectures ont rencontré de nombreux succès pour résoudre des tâches de génération de langage complexes, telles que la modélisation du langage [195, 264], la traduction automatique [181, 265], la reconnaissance vocale [239, 113], ou la génération de légendes sur des images [137]. .

L'apprentissage transféré pour les systèmes de TAL. En 2017, une seconde révolution s'est mise en marche avec le développement d'une nouvelle architecture de réseaux de neurones pour les systèmes de TAL, le Transformer [277]. Par la manière dont ils modélisent les dépendances globales et locales de données séquentielles, l'utilisation des Tranformers est moins coûteuse en temps de calcul que celle des RNN, et ils ont ainsi permis l'essor de réseaux avec des milliards de paramètres qui permettent de modéliser des gigantesques corpus de textes. A partir de 2019, des systèmes de langage pré-entraînés génériques, basés sur des Transformers, ont vu le jour, tels que BERT [65], GPT [230] ou T5 [233]. Contenant des milliards de paramètres, entraînés sur des gigantesques jeux de données textuels provenant de sites internet, ils encodent des représentations du langage universelles. De tels systèmes ont permis l'essor de l'*apprentissage transféré* dans le domaine du TAL, le procédé de partir d'un modèle pré-entraîné appris sur une tâche auxiliaire, et de l'adapter en aval sur une tâche cible. Aujourd'hui, de nombreux systèmes de TAL sont perfectionnés à partir de modèles de langage pré-entraînés, ce qui (i) accélère la phase d'entraînement, (ii) améliore généralement les performances sur la tâche cible (surtout sur de petits jeux de données) et (iii) permet un transfert plus rapide sur une nouvelle tâche et un nouveau jeu de données.

Représenter le langage naturel avec des modèles génératifs profonds. Bien que les systèmes de langage pré-entraînés tentent d'encoder une représentation universelle du langage et peuvent être adaptés en aval à de nombreuses tâches de TAL, ils ne parviennent cependant pas à modéliser avec fidélité la complexité et la diversité qui caractérisent le langage naturel [170, 127, 272]. Une alternative à ces gigantesques réseaux de neurones déterministes est de représenter n'importe quel jeu de données textuel comme un ensemble fini d'échantillons d'une distribution de probabilité sous-jacente. C'est l'objectif des *modèles génératifs*, qui visent à estimer la distribution de la donnée, étant donné l'accès à un jeu de données. Depuis quelques années, les modèles génératifs basés sur des réseaux de neurones, appelés modèles génératifs profonds (DGM), ont suscité beaucoup d'intêret dans la communauté scientifique spécialiste en apprentissage profond. Des modèles comme l'auto-encodeur variationnel (VAE) [143] ou les réseaux adversariaux génératifs (GAN) [106] ont montré des résultats impressionnants dans le domaine du traitement de l'image. Ces approches - en particulier les VAE - ont été utilisées ensuite sur des

problèmes de TAL[49, 90, 177, 167], mais développer des modèles génératifs profonds adaptés à de la donnée séquentielle demeure un problème de recherche à part entière [35, 30]. Dans ce cadre, de tels modèles, en encodant des observations séquentielles avec une riche structure latente, pourraient mieux modéliser la diversité présente dans le langage humain, ou pour des problèmes de prévision de séries temporelles, pourraient prendre en compte le bruit et l'aléa des données de mesure, et prédire des estimations accompagnées d'intervalles de confiance. Une approche intéressante - mais à ce jour à peine explorée - pour apprendre des DGM sur de la donnée séquentielle serait d'utiliser des méthodes d'échantillonnage (comme les algorithmes de MCMC et SMC [179]) pour estimer la distribution inconnue d'un DGM (par opposition aux algorithmes d'Inférence Variationnelle [24], qui forment le coeur de l'entraînement des VAE).

La modélisation du langage. Au sein de cette thèse, nous nous intéressons à une sous-branche des tâches de TAL, les Modèles de Langage (LM). La modélisation du langage est au coeur des tâches de génération de langage, qui visent à produire du langage naturel pour une application spécifique (e.g la traduction automatique, le résumé de texte, la génération de dialogue, la génération de légendes sur une image ou une vidéo, la génération de paraphrases, etc). Un modèle de langage apprend une distribution de probabilité sur des mots à partir d'un vocabulaire (en général constitué de plusieurs dizaines de milliers de mots). On distingue les Modèles de Langage inconditionnels, qui apprennent simplement une distribution sur les mots, et les Modèles de Langage conditionnels, qui apprennent une distribution sur des mots étant donné un contexte. Le contexte peut être une phrase d'entrée (pour les systèmes de traduction automatique ou de génération de paraphrases), un paragraphe ou un document (pour les systèmes de résumé de texte), une image (pour les systèmes de génération de légendes), une vidéo, une base de données, etc.

Entrainer des Modèles de Langage par apprentissage supervisé. Il existe deux principales approches pour entraîner un Modèle de Langage. La plus utilisée repose sur de l'apprentissage supervisé (SL). Pour entrainer un Modèle de Langage, on apprend en pratique une distribution $p_\theta(w_t|w_0, ..., w_{t-1})$, dépendant de paramètres θ à estimer, sur le vocabulaire $\mathcal{V}$ qui vise à prédire le prochain mot w_t sachant les mots précédents $(w_0, ..., w_{t-1})$. Apprendre un Modèle de Langage peut être donc assimilé à un problème de classification séquentiel, avec le prochain mot w_t qui est la vraie étiquette, et les mots passés qui constituent la donnée d'entrée. Avec cette approche, les paramètres θ du modèle de langage sont mis à jour par descente de gradient, en minimisant une fonction de coût qui est l'entropie croisée [195]. Les grands Modèles de Langage pré-entraînés [230, 232, 33] sont typiquement entraînés comme des Modèles de Langage inconditionnels sur de gigantesques corpus de textes, en utilisant l'apprentissage supervisé décrit précédemment. Pour les adapter en aval à une tâche cible (dans l'objectif de créer un Modèle de Langage conditionnel), on utilise de nouveau des méthodes d'apprentissage supervisé : à partir des paramètres de pré-entraînement du Modèle de Langage générique, les paramètres sont mis à jour sur la tâche et le dataset cibles,

en utilisant une fonction de coût issue de l'apprentissage supervisé.

Bien que relativement faciles à entrainer et à implémenter, les Modèles de Langage basés sur de l'apprentissage supervisé ont plusieurs défauts. Premièrement, le problème-clé quand on entraine des Modèles de Langage est la génération de langage à l'inférence, i.e écrire du texte avec celui-ci une fois entraîné. A partir d'une séquence de mots ou simplement d'un contexte pour les Modèles de Langage conditionnels, le Modèle de Langage génère une séquence de mots, en produisant de manière successive un nouveau mot, qui est ensuite ajouté à la séquence de mots d'entrée du modèle, tel qu'illustré sur la Figure 2.1. Nous remarquons ainsi qu'il y a une divergence entre l'entraînement d'un LM, dont la fonction de score maximise la vraisemblance d'un nouveau mot sachant l'état courant, et l'inférence, où les vrais mots cibles ne sont plus disponibles, et sont remplacés par les mots produits successivement par le Modèle de Langage. Ce phénomène est connu sous le nom de "biais d'exposition" [18], et peut provoquer des problèmes de cohérence dans la génération du texte à l'inférence. Au fur et à mesure que la séquence de mots générée s'allonge, les fautes peuvent s'accumuler le long de la phrase, créant du texte incohérent et incorrect. Par ailleurs, les Modèles de Langage appris par apprentissage supervisé ont tendance à être sur-confiants, i.e à présenter une distribution "piquée" qui les fait sélectionner seulement quelques mots dans le vocabulaire, et générer des séquences de mots redondantes. L'utilisation de méthodes de décodage de texte plus ou moins sophistiquées atténue ce problème, mais générer du langage à l'inférence qui produit du texte à la fois de bonne qualité et diversifié reste un problème de recherche ouvert [127]. Deuxièmement, entraîner des Modèles de Langage conditionnels avec de l'apprentissage supervisé nécessite de gros jeux de données étiquetés à la main. En plus d'être un procédé coûteux, la dépendance à des jeux de données supervisés produit également des modèles qui ont tendance à hériter des biais naturellement présents dans cette donnée. Ce problème a été mis en avant même sur les Modèles de Langage génériques pré-entraînés, qui peuvent générer du texte raciste et insultant [98], avoir des performances médiocres quand ils utilisent des dialectes minoritaires [26, 148, 302], et peuvent être piégés par des attaques adversariales [135, 279].

Apprendre des Modèles de Langage par Apprentissage par Renforcement. L'apprentissage d'un Modèle de Langage et la génération de texte peuvent être également interprétés comme un problème de décision séquentiel. L'agent computationnel, étant donné un contexte et un historique de mots déjà prononcés, doit décider quel prochain mot prononcer. L'apprentissage par renforcement (RL) [266] regroupe un ensemble d'algorithmes qui apprennent des comportements optimaux (appelés "politiques") étant donné un problème de décision séquentiel. En apprentissage par renforcement, un agent interagit avec un environnement, en choisissant successivement des actions qui lui font changer d'état, et pour lesquelles il reçoit un signal scalaire, appelé récompense. Dans le cadre de la génération de langage, l'agent est le modèle de langage conditionnel à apprendre. Son état à un pas de temps t du processus de décision est la séquence de mots qu'il a déjà prononcés $w_{1:t-1} = w_{<t}$ et le contexte c. Sa prochaine action est le prochain mot w_t qu'il doit prononcer. La récompense reçue peut être une valeur issue d'une métrique

de langage, ou simplement un score évaluant sa performance sur la tâche de TAL à résoudre. Les modèles de langage basés sur de l'apprentissage par renforcement sont séduisants pour plusieurs raisons. Premièrement, leur apprentissage imite plus fidèlement la manière dont les humains apprennent une langue. En effet, les enfants apprennent à parler via une interaction avec leur entourage [118], et par l'intermédiaire d'un renforcement continu [34]. Une sous-branche du TAL se concentrant sur l'émergence du langage chez les êtres humains le modélise sous la forme d'un problème d'apprentissage par renforcement [155, 42, 197]. Par ailleurs, la théorie de la concrétisation dans les sciences cognitives stipule que notre raisonnement, notre langage et notre pensée sont inextricablement façonnés par notre perception et nos actions [17, 287]. Notre compréhension du monde viendrait de deux éléments : notre connexion sensorielle à travers nos cinq sens, et notre interaction avec la réalité physique et des personnes extérieures. Le dialogue basé sur des objets visuels [262, 52] suit ce paradigme, en apprenant le langage à travers un cadre de RL interactif, en l'ancrant avec de la perception visuelle. Deuxièmement, de tels Modèles de Langage permettent plus de flexibilité que les LM entraînés par apprentissage supervisé. Dans le cadre du RL, le processus d'apprentissage de l'agent est perfectionné par une récompense, qui peut être n'importe quelle quantité scalaire. Ce peut être notamment une fonction non différentiable ; en pratique, les LM basés sur du renforcement peuvent directement optimiser des métriques de langage à l'échelle de la phrase (e.g celles utilisées pour évaluer les modèles de langage une fois entraînés). Par exemple, [219, 161] ont développé des algorithmes de RL avec des récompenses basées sur de telles métriques pour entraîner des systèmes de résumé de texte,[238, 217] pour la génération de légende sur une image ou sur une vidéo, ou encore [173] pour créer un système de génération de paraphrases. Concrètement, de tels modèles souffrent généralement moins des problèmes de dégénération de texte mentionnés précédemment dans le cadre de l'apprentissage supervisé. Ces modèles permettent aussi plus de flexibilité dans la collection des jeux de données tout en générant naturellement du langage plus diversifié, du fait qu'ils ne reposent pas sur des exemples supervisés. Enfin, en associant des états et actions à un environnement concret, ils sont plus facilement interprétables. Historiquement, les modèles de langage basés sur du RL viennent du domaine des systèmes de dialogue à tâches [162, 223, 299, 258], où le gestionnaire de dialogue est appris par renforcement. Depuis, des dialogues à tâches plus modernes [55, 51, 164, 205] se sont développés, et utilisent de l'apprentissage par renforcement profond (i.e des algorithmes de RL avec des estimateurs basés sur des réseaux de neurones). Les jeux de données GuessWhat ? ! [55, 262] et Visual Dialog [51] ont créé des jeux de dialogue où est le but est de trouver le bon objet dans une image en posant des questions fermées, alors que le jeu de données Deal or No Deal ? [164] est un jeu de dialogue visant à répartir des objets de différentes valeurs entre deux négociateurs. Les jeux vidéos textuels [205] sont d'autres benchmarks pour les systèmes de TAL à base de RL (par exemple, le jeu Zork [23]). Ces jeux décrivent l'environnement au joueur à travers du langage naturel, et lui-même interagit dans le jeu via du texte (pour par exemple récupérer des objets dans une pièce, changer de lieu, etc...).

Bien que séduisants et prometteurs, la recherche sur les modèles de langage appris par renforcement n'en est qu'à ses débuts, et plusieurs problématiques restent encore à résoudre pour utiliser de telles approches sur des

tâches de génération de langage complexes, et obtenir des performances comparables aux systèmes entraînés de manière supervisée. Premièrement, il y a des défis inhérents à l'apprentissage par renforcement profond, qui est très gourmand en nombre d'échantillons pour obtenir de bonnes performances, qui souffre de grande variabilité et parfois d'instabilité, car de tels algorithmes ne bénéficient pas de garanties théoriques de convergence [15, 19, 276]. Lorsque que l'on applique de l'apprentissage par renforcement au texte, l'une des problématiques majeures est l'exploration d'un espace d'actions constitué de plusieurs dizaines de milliers de mots, de manière à choisir les mots déclenchant une récompense positive. L'exploration dans un large espace d'actions discret est un problème-clé pour faire passer à l'échelle les algorithmes de RL à des cas de la vie réelle, mais demeure un problème de recherche ouvert [273]. Les systèmes de langage à base de RL existants évitent le problème, soit en incluant une phase de pré-entraînement supervisé pour l'agent qui représente le modèle de langage [262, 52], soit en utilisant des objectifs hybrides mélant RL et SL [238], ou encore en apprenant un langage restreint à un petit vocabulaire, et à des phrases très courtes [296].

1.2 Problématiques de recherche

Dans ce document, nous nous intéressons à l'amélioration des systèmes de génération de langage, à la fois du point de vue des modèles, et du point de vue de l'apprentissage (i.e la manière dont les Modèles de Langage sont entraînés). Nous proposons tout d'abord de pallier les limites des réseaux de neurones déterministes de grande dimension par l'intermédiaire de nouveaux modèles génératifs. Nous explorons également de nouvelles pistes afin de résoudre les limites des Modèles de Langage appris de manière supervisée, et par apprentissage par renforcement. Nous considérons précisément les problématiques de recherche suivantes.

1. Comment proposer des modèles génératifs à données latentes tirant partie des mécanismes d'attention introduits par les modèles Transformers ? Puis comment utiliser les méthodes de Monte Carlo Séquentielles pour l'apprentissage de telles architectures de réseaux de neurones, dans le but de développer des modèles génératifs profonds adaptés à de la donnée séquentielle ?

2. Comment développer des méthodes de Monte Carlo Séquentielles pour estimer des modèles à espace d'états quand la dynamique des états latents ou la vraisemblance conditionnelle des observations ne peuvent être évaluées, ce qui est le cas de nombreux modèles génératifs profonds ? Comment faire cela tout en réduisant significativement le temps de calcul des solutions numériques existantes ?

3. Comment améliorer la génération de langage des Modèles de Langage supervisés, en particulier améliorer la diversité du texte généré ?

4. Comment résoudre le problème d'exploration de l'espace d'actions pour les Modèles de Langage appris par RL, de manière à ce qu'ils puissent être appris "ex nihilo", tout en généralisant à de grands vocabulaires ?

Modèles génératifs profonds basés sur des méthodes de Monte Carlo Séquentielles. Dans le contexte de la modélisation du langage, on s'intéresse à développer des réseaux de neurones qui représenteraient la nature stochastique du langage à travers des états latents non observés, contrairement aux architectures classiques qui utilisent seulement une couche softmax pour représenter la distribution du LM. Plusieurs travaux ont essayé d'adapter les modèles génératifs profonds existants à des tâches de génération de langage, tels que les VAE [49, 76, 90, 167] ou les GAN [84, 251, 54]. Cependant, les premiers souffrent du problème de "l'effondrement de la loi a posteriori" (les faisant dégénerer vers des réseaux déterministes), alors que les derniers sont connus pour être mal adaptés à la génération de texte à l'inférence [30, 35]. Les GAN présentent en effet un phénomène d'"effondrement de modes", i.e le modèle n'apprend que quelques modes de la distribution, générant en pratique que quelques phrases différentes. Dans ce document, nous choisissons d'utiliser des Méthodes de Monte Carlo Séquentielles (SMC) pour estimer la distribution des états latents d'un DGM étant donné une séquence d'observations, ce qui permet de modéliser des distributions latentes complexes. Les Méthodes de Monte Carlo Séquentielles ont été utilisées au sein de réseaux de neurones seulement dans quelques travaux de recherche, et toujours dans un schéma d'apprentissage basé sur de l'Inférence Variationnelle [203, 191, 198, 189]. Nos travaux de recherche proposent au contraire d'estimer directement la log-vraisemblance des observations d'un DGM par des méthodes SMC. Nous développons d'abord un modèle génératif basé sur un réseau Transformer, dont les états latents et la fonction de score sont estimés via un simple algorithme de lissage. Ce modèle génératif profond peut s'appliquer à un cadre plus large que celui des Modèles de Langage, et permet notamment de représenter l'incertitude dans les problèmes de données séquentielles. Pour commencer, on l'applique donc sur des tâches de quantification d'incertitude pour la prévision de séries temporelles. Cependant, utiliser des méthodes de SMC, caractérisées par un coût de calcul élevé, sur de larges réseaux de neurones est un problème de recherche en soi. Par conséquent, dans un second temps, nous développons un algorithme de lissage plus efficace, dont le coût de calcul réduit le rend plus à même d'être utilisé sur des architectures d'apprentissage profond.

Apprentissage par renforcement tronqué pour espace d'actions de langage. La quatrième problématique de recherche a été à peine explorée par la communauté. Dulac-Arnold et al., Tennenholtz et al, Chandak et al [73, 273, 43] ont developpé des algorithmes génériques pour de large espaces d'actions discrets, qui les transforment en espace continus, à partir desquels des algorithmes classiques de contrôle continu sont utilisés pour apprendre une politique sur de tels espaces. Cependant, ces algorithmes ne prennent pas en compte la spécificité du Langage Naturel, ce qui les rend mal adaptés pour passer à l'échelle sur des tâches de langage complexes. Zahavy et al, Seurin et alZahavy et al. [300], Seurin et al. [254] ont proposé des algorithmes de "Q-learning" incorporant un signal éliminatoire pour éliminer des actions interdites, ce qui permet progressivement de réduire la taille de l'espace d'actions effectif. Cependant, l'élimination d'actions sur un espace de vocabulaire de plusieurs dizaines de milliers de mots est un processus qui reste coûteux. De plus, l'emploi d'algorithmes de "Q-learning" (mal adaptés

pour de grands espaces d'actions) exacerbe la difficulté de la méthode. Dans nos travaux, nous proposons d'utiliser spécifiquement la structure du langage pour tronquer l'espace d'actions de langage à un espace de mots de taille raisonnable. Pour modéliser la structure du langage dans un cadre d'apprentissage de bout-en-bout, nous utilisons la représentation linguistique d'un modèle de langage pré-entraîné. D'un point de vue linguistique, cela revient à fournir a priori une connaissance générique du langage à l'agent de renforcement qui représente le Modèle de Langage, qui apprend ensuite par RL le pragmatisme spécifique à la tâche. D'un point de vue de la modélisation, l'approche proposée combine de manière simple un Modèle de Langage basé sur du RL avec des systèmes de langage pré-entraînés. La méthode proposée évite notamment les défis numériques posés par le ré-entraînement d'un Modèle de Langage pré-entraîné générique avec du RL, dont les difficultés proviennent à la fois de l'instabilité des systèmes de RL profond d'une part, et de l'instabilité résultant du ré-entraînement de modèles à milliards de paramètres d'autre part [65, 160, 199, 233, 45]. En outre, les Transformers ont empiriquement tendance à présenter de mauvaises performances quand ils sont entraînés par RL [215].

1.3 Contenu de la thèse

Les chapitres 3 et 4 présentent l'état de l'art associé aux Modèles de Langage basés sur de l'apprentissage profond. Les chapitres 5, 6 et 7 regroupent les travaux de recherche développés durant cette thèse.

Dans le chapitre 3, nous décrivons l'état de l'art sur les réseaux de neurones pour de la donnée séquentielle. Nous commençons d'abord par les principes élémentaires de l'apprentissage profond, puis nous nous concentrons ensuite sur les architectures de réseaux de neurones spécifiques à des problèmes de données séquentielles, des réseaux récurrents aux Transformers. Enfin, nous présentons les modèles génératifs pour des problèmes de génération de séquences, en détaillant dans la dernière section les méthodes de Monte Carlo Séquentielles, des algorithmes d'échantillonnage qui permettent notamment d'approcher la fonction de score non explicite dans de tels modèles génératifs.

Dans le chapitre 4, nous décrivons précisément le problème de la modélisation du langage, et de la génération de texte. Nous présentons ensuite le cadre théorique pour apprendre des Modèles de Langage par apprentissage supervisé, puis par apprentissage par renforcement. Enfin, nous présentons les métriques d'évaluation qui permettent de mesurer la qualité et la performance d'un Modèle de Langage.

Le chapitre 5 présente TrufLL, pour Apprentissage par Renforcement tronqué pour le Langage. TrufLL développe un cadre d'apprentissage par renforcement pour apprendre des Modèles de Langage par RL "ex-nihilo", en tronquant l'espace d'actions du vocabulaire en utilisant un Modèle de langage pré-entraîné. L'algorithme est appliqué à deux tâches de génération de questions visuelles. A partir d'une paire (image, réponse), l'agent de RL doit générer une question sur l'image qui correspond à la réponse fournie. Le chapitre est completé dans l'annexe A, qui fournit plus de détails sur les hyper-paramètres des modèles évalués, présente des résultats d'expérience supplémen-

taires, détaille le protocole de l'évaluation humaine, et fournit des échantillons de texte générés par les différents modèles supplémentaires.

Le chapitre 6 introduit le Sequential Monte Carlo (SMC) Transformer, un modèle génératif profond pour des problèmes de prévision séquentiels, basé sur un modèle d'auto-attention stochastique. Dans un cadre de régression, le SMC Transformer prédit une distribution d'observations, au lieu d'une simple estimation ponctuelle. Dans un cadre de classification, le modèle permet de représenter la variabilité des observations d'entrée par l'intermédiaire d'états stochastiques latents. Nous évaluons tout d'abord le SMC Transformer sur des problèmes de prévision de séries temporelles à plusieurs pas de temps, et montrons empiriquement sa capacité à prédire correctement l'incertitude prédictive, contrairement aux réseaux de neurones bayésiens couramment utilisés. Dans un second temps, nous l'appliquons sur une tâche de modélisation de langage, où nous montrons les défis numériques restants pour faire passer à l'échelle un tel modèle sur des tâches de génération de langage complexes. Le chapitre est accompagné de l'annexe B, qui fournit plus de détails sur les hyper-paramètres des différents modèles et sur les jeux de données de séries temporelles, et présente des résultats d'expérience supplémentaires.

Le chapitre 7 présente l'algorithme "Backward Importance Sampling" (BIS), un nouvel algorithme de lissage en ligne pour estimer la log-vraisemblance de modèles à espace d'états complexes. BIS a un coût computationnel largement réduit par rapport à l'état de l'art, tout en étant applicable à une plus grande classe de modèles à espace d'états. L'algorithme est évalué pour l'estimation de paramètres et l'estimation d'états de plusieurs modèles, en particulier sur un réseau récurrent stochastique, et un modèle Lokta-Volterra stochastique multi-varié, pour laquelle la densité de transition n'est pas connue, et est estimée via un cadre de pseudo-marginalisation. Le chapitre est accompagné de l'annexe C, qui détaille le cadre théorique permettant d'estimer la densité de transition des équations différentielles stochastiques considérées ainsi que de leur gradient afin de mettre en place des algorithmes de maximum de vraisemblance récursifs.

Le chapitre 8 conclut ce document, en résumant les travaux de recherche et les principales contributions, et en listant de potentielles directions futures de recherche pour approfondir les problématiques traitées au cours de cette thèse.

1.4 Contributions

Les contributions de cette thèse sont les suivantes.

— Dans le chapitre 5, nous proposons le premier algorithme d'apprentissage par renforcement pour entraîner des Modèles de Langage conditionnels "ex-nihilo" (i.e sans phase de pré-entraînement supervisée), qui s'applique à de grands vocabulaires (comprenant environ 15,000 mots) et qui atteint des performances proches de Modèles de Langage basés sur du RL qui incluent une phase de pré-entraînement supervisée.

  i) L'algorithme propose une nouvelle manière de combiner des Modèles de Langage conditionnels appris

par renforcement à de grands Modèles de Langage pré-entraînés, par l'intermédiaire d'un mécanisme de troncation pour l'espace d'actions constitué de langage naturel, voir Section 5.3.

ii) Cet algorithme constitue une première étape pour s'affranchir de jeux de données étiquetés à la main pour apprendre des Modèles de Langage conditionnels, tout en présentant une distribution de langage plus variée que les modèles incluant une phase d'entraînement par maximum de vraisemblance, voir Section 5.5.

— Dans le chapitre 6, nous proposons le SMC Transformer, nouveau modèle à base de réseaux de neurones Transformer stochastiques, accompagné de méthodes de Monte Carlo Séquentielles pour estimer directement la fonction de score par l'intermédiaire de l'identité de Fisher, voir la Section 6.3 et le Lemme 1.

i) Nous montrons empiriquement que le modèle estime correctement la variabilité des observations pour des modèles synthétiques à bruit d'observations connu, contrairement aux approches couramment utilisées pour la quantification d'incertitude, tels que les réseaux de neurones bayésiens et les approches de type Monte Carlo dropout, voir Section 6.4.2.

ii) Finalement, dans le cadre de prévisions de séries temporelles sur des données réelles, nous montrons empiriquement que le SMC Transformer atteint de meilleures performances sur des métriques de quantification d'incertitude usuelles que les approches classiques précédemment citées, voir Section 6.4.3.

— Dans le chapitre 7, nous proposons un nouvel algorithme de lissage pour l'estimation en ligne des modèles à espace d'états, voir Section 7.4.

i) L'algorithme développé a un coût computationnel bien moindre que les alternatives de l'état de l'art, tout en présentant des performances similaires. Il s'agit donc d'un premier pas pour faire passer à l'échelle les techniques de lissage et les appliquer à des architectures d'apprentissage profond. Nous l'appliquons notamment à l'estimation d'état d'un réseau récurrent stochastique, voir Section 7.5.1.

ii) L'algorithme s'applique à un ensemble plus vaste de modèles à espace d'états que l'état de l'art : l'utilisation de techniques de pseudo-marginalisation le rend applicable à des modèles dont la densité de transition et/ou la densité conditionnelle des observations sont inconnues. Dans ce cadre, nous l'appliquons tout d'abord à l'estimation en ligne des paramètres d'un modèle sinus (Sections 7.5.2 et 7.5.3), pour lequel nous utilisons un estimateur de la densité de transition et de son gradient basés sur un estimateur de Poisson Généralisé, voir Proposition 1 et Proposition 2 dans l'Appendice C.1. Finalement, nous l'évaluons sur un processus de diffusion multi-dimensionnel, pour lequel nous combinons des estimateurs paramétriques [6] avec l'astuce de Wald [80], afin d'obtenir une estimation de la densité de transition non biaisée et positive presque sûrement, voir Section 7.5.4 et Appendice C.1.2.

Chapitre 2

Introduction (English Version)

2.1 Motivation and Context

Language is an essential component of numerous artificial intelligence (AI) systems. Central to human thought, it allows to encode abstractions, to generalize over concepts and objects, to communicate intentions and needs, both to ourselves and to other parties [108]. Developing systems that can understand and generate human language is thus one of the main challenge on the quest towards artificial intelligence. This is the aim of Natural Language Processing (NLP), which lies at the intersection between linguistics, computer science, and artificial intelligence. Today, the variety of NLP systems reflects the ubiquity of Language and its centrality in assisting human intelligence. Industrial NLP applications include text completion, web search, automated translation, sentiment analysis (for marketing, detecting online harmful behaviors, ...), online advertisement matching, automatic mining of legal and medical reports, chatbots, and many others.

Natural Language Processing. At the core of any NLP task, lies the important issue of natural language understanding (NLU) [48]. NLU involves three key challenges : understanding the thought process, understanding the representation and meaning of the linguistic input, and understanding the world knowledge. In NLP, we thus distinguish the notion of *syntax* that deals with the grammar and structure of sentences, *semantics* that deals with the meaning of words and sentences, *discourse* that deals with the structure of different kinds of text using document structure, and *pragmatics* that deals with the knowledge that comes from the outside world.

Deep Learning and NLP. From the years 1990s, the NLP field has been dominated by the Empiricist approach, which assumes that humans learn the detailed structure of Language through experience, and has developed NLP systems based on statistical learning methods applied to large amounts of text [1, 263, 86, 117]. In the years 2010s, the performance and versatility of statistical NLP systems were boosted by the emergence of Deep Lear-

ning [294, 107], and neural networks specialized for processing text were developed. Recurrent Neural Networks (RNN) architectures [74, 125, 46], based on the idea of processing sequential information, can easily model texts of variable lengths, including very long sentences, paragraphs and documents. They have showcased impressive successes on complex natural language generation (NLG) tasks, such as Language Modelling [195, 264], Machine Translation [181, 265], Speech Recognition [239, 113] or Image Captioning [137].

Transfer Learning in NLP. In 2017, a second revolution came with the development of a new neural network architecture for NLP systems, the Transformer [277]. From the way they model local and global dependencies in sequential data, Transformers are more computational efficient than RNN, and have enabled billions parameters networks, that scale to very large corpus of text. From 2019, using Transformer-based networks, generic pretrained Language Systems have been developed such as BERT [65], GPT [230], or T5 [233]. Made of billions of parameters, trained on large-scale web-scraped datasets containing several terabytes of data, they encode highly generic language representations. They have hence enabled *transfer learning* for NLP systems, the process of using a pretrained model learned on a surrogate task, and adapting it to a downstream task of interest. Today many state-the-art NLP systems are fine-tuned from a pretrained Language Model, which (i) accelerates their training phase, (ii) generally improves the overall performance on the downstream task (even for small-size text datasets) and (iii) allows faster adaptation to a new task and dataset.

Representing Natural Language with Deep Generative Models. While pretrained Language systems attempt to encode universal language representations, they are nonetheless falling short in modelling the complexity and diversity found in Natural Language [170, 127, 272]. One alternative to such large-scale deterministic networks is to view any kind of observed text dataset as a finite set of samples from an underlying distribution. This is the purpose of *Generative Models*, which aim to approximate this data distribution given access to the dataset. For a few years, Generative Models based on Neural Networks - called Deep Generative Models (DGM) - have been a major topic of interest in the Machine Learning research community, with models like Variationnal Auto-encoders (VAE) [143] and Generative Adversarial Networks (GAN) [106] showcasing impressive results in the Computer Vision field [235, 138]. Such approaches - in particular VAE - have been used to solve NLP problems [49, 90, 177, 167], but designing DGM with strong performances on sequence modelling tasks remains an open research problem [35, 30]. In this setting, by encoding sequential observations with a rich latent structure, such models could represent the diversity naturally found in Human Language. In other sequential tasks such as time time-series forecasting, it could account for the noise and randomness present in data from measurements, and output estimations with confidence intervals. One interesting - yet barely explored - approach for learning DGM for sequential data is the use of sampling-based methods (such as MCMC and SMC algorithms [179]) to estimate the true (intractable) underlying distribution of the Generative Model (by opposition to Variational Inference algorithms [24], which form the basis of VAE training).

Language Modelling. In this thesis, within Natural Language Processing, we focus on Language Modelling tasks. Language Modelling is at the core of Natural Language Generation tasks, that focus on producing natural language for a specific application (e.g automatic translation, text summarization, dialog generation, image or video captioning, paraphrase generation, etc). A Language Model (LM) learns a probability distribution on words from a vocabulary $\mathcal{V}$ (usually made of thousands of words). We distinguish unconditional language models, that learn simply a distribution over words, and conditional language models that learn a distribution over words given a context. The context could be an input sentence (for Machine Translation or Paraphrase Generation systems), an input paragraph or document (for Text Summarization), an image (for Image Captioning or Visual Question Generation systems), a video, a database, and so on.

Learning Language Models with Supervised Learning. There are two modern approaches to train Language Models. The most popular approach relies on Supervised Learning (SL). Indeed, we learn a probability distribution $p_\theta(w_t|w_{0:t-1}, c)$ on $\mathcal{V}$ that aims at predicting a next word w_t given a sequence of past words $w_{0:t-1}$, and an (optional) context c, where $w_{0:t-1}$ stands for the past sequence of words $(w_0, \ldots, w_{t-1})$. Learning Language Models can be viewed as a sequential classification problem, the next token w_t being the ground truth label and the past sequence of words the input data. The parameters of the language model are then updated with gradient-based learning using a cross-entropy loss [195]. Pretrained generic Language Models [230, 232, 33] are typically pretrained as unconditional language models on large-scale corpus of text using a supervised learning objective. Adapting them to a downstream task of interest (usually to build a conditional language model) again uses supervised learning : from the pretraining parameters of the pretrained Language Model, the parameters are updated on the downstream task using a SL objective.

While relatively easy to train and implement, SL-based Language Models suffer from several pitfalls. First, the key challenge when learning Language Models is Natural Language Generation at inference, i.e writing text with the trained LM. From an input sequence of words or simply the context for conditional language models, the LM generates a new sequence of words, by successively producing a word, which is then added to the input sequence of past words that serve to produce the next one, as illustrated in Figure 2.1. We notice that there is thus a discrepancy between training, whose objective maximizes the likelihood of the next word given the current state, and inference, where true target words are unavailable and are replaced by the words generated by the Language Model itself. This is known as the exposure bias [18], and triggers text generation issues at inference. As the generated sentence grows longer, errors can accumulate quickly along the sentence, creating incorrect and incoherent text. Additionally, Language Models learned with Supervised Learning tend to be overconfident, i.e having a very peaky distribution that make them selecting only a few words. In practice, at inference, this means that the generated text might include repetitive sequence of words. Different text decoding techniques might mitigate this issue, but generating language at inference that produces both high quality and diverse text samples remains an open research problem [127]. Se-

condly, training conditional language models with supervised learning requires hand-labelled large-scale datasets. Besides being a costly process, the reliance on supervised datasets also produces models that might inherit the language biases present in those datasets. This problem has been pointed out even on generic pretrained Language Models, which are prone to generate toxic text [98], or underperform when referring to minority groups [26, 148, 302], and can be tricked with adversarial attacks [135, 279].

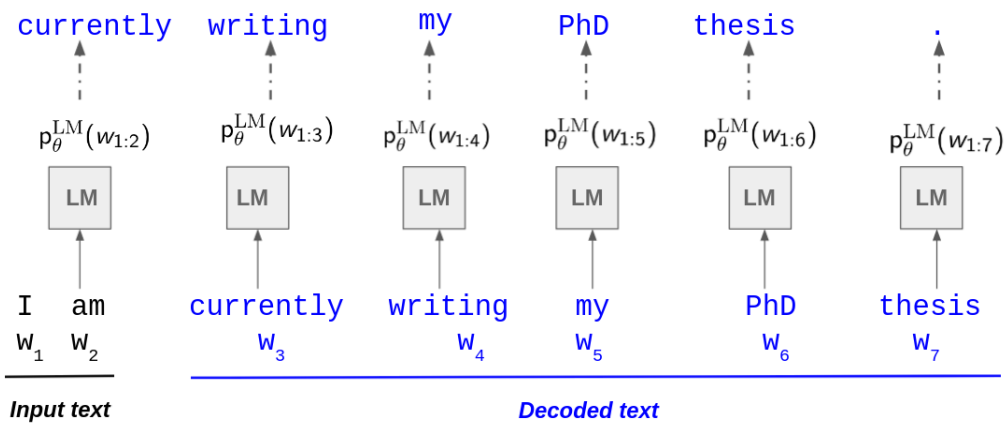


FIGURE 2.1 – Illustration of the decoding process for generating text with a trained Language Model. From an input text "I am", the language model decodes sequentially the sequence "currently writing my PhD thesis." At timestep t of the decoding process, a new word w_t is generated from the probability distribution $p_\theta^{\text{LM}}(w_{1:t-1})$. The decoded word is then added to the input sequence of the language model $w_{1:t}$ that generates the next decoded word.

Learning Language Models with Reinforcement Learning. Language Modelling can be also seen as a sequential decision making problem. The computational agent, given a context and a past history of words, has to decide which word to be uttered next. Reinforcement Learning (RL) [266] approaches are methods that learn optimal behaviors (referred to as policies) given a sequential decision making problem. In RL, an agent interacts with an environment, by taking successive actions and moving into states, and receiving a scalar feedback, called the reward. In a Language Generation setting, the Agent is the conditional Language Model to be learned. Its state at a given time step t is the past sequence of words $w_{1:t-1} = w_{<t}$ it already uttered and the context c. Its next action is the next word w_t to be uttered. The reward received could be a language metric evaluating the sequence of words being uttered, or simply a task score.

RL-based Language Models are appealing for several reasons. First, they more closely mimick the way humans learn language. Indeed, children learn language via interaction with their surroundings [118] and through continuous feedback and reinforcement [34]. A subfield of NLP focuses on studying the emergence of language [155, 42, 197], and frames it as a Reinforcement Learning problem. Additionally, the embodiment theory in cognitive science states that our reasoning, language, and thoughts are inextricably shaped by our perception and actions [17, 287]. In other words, our understanding of the world would come from two key components : our sensorimotor connection through our five senses and our interaction with the physical reality and other people. Visual Grounded Dialog [262,

52] follows this paradigm, by learning Language through a RL interactive framework, and grounding it with visual perception.

Secondly, they allow more flexibility than SL-based Language Models. In a RL setting, the learning process of the computational agent producing language is reinforced by a reward, which can be any scalar quantity. This can be a non-differentiable function, which means in practice, that RL-based Language Models can directly optimize sentence-level language metrics (e.g the ones used at test time for evaluating the Language Models), unlike their SL counterparts. For instance, [219, 161] use a reinforcement learning algorithm with such sentence-level metrics to train text summarization systems, [238] to train an image captioning model, [217] for performing video captioning, and [173] to build a paraphrase generation model. Concretely, this results in avoiding the text degeneration issues mentioned previously such as the exposure bias, or the overconfidence of LM in selecting only a few words.

By not relying directly on ground-truth text samples, they allow also more flexibility in data collection, while naturally generating more diverse text samples at inference. Finally, by mapping states and actions to a concrete environment, they are more easily interpretable.

Historically, RL-based Language systems come from the field of task-oriented dialog systems [162, 223, 299, 258], where the dialogue manager is learned through interaction. Since then, as illustrated in Figure 2.2, modern word-based task-oriented dialogues [55, 51, 164, 205] have been developed and use deep Reinforcement Learning methods (i.e RL with neural network function approximators). The GuessWhat ?! [55, 262] and Visual Dialog [51] datasets develop a dialog game to find the correct object in an image by asking questions, while the Deal or No Deal ? Dataset [164] develops a Dialog game to split a set of valued items between two negociators. Text-based computer games [205] are other test-beds for RL-based NLP systems (e.g the Zork game [23]). Such games describe their world to the player through natural language and expect the player to interact with the game using text (for instance to grab objects in a room, move to the next room, etc).

While appealing and promising, research on RL-based Language Models is at its infancy, and several challenges remain to be solved for scaling them to complex NLG tasks, and achieve performances comparable to SL ones. First, there are inherent challenges when training Deep Reinforcement Learning agents : Deep RL suffers from high-variance, sample inefficiency, sometimes instability as there are no theoretical guarantees of convergence when combining RL with neural networks function approximations [15, 19, 276]. When applied in a NLP setting, one major challenge is how to explore a Natural Language Action Space made of over ten thousands words, so that words triggering a positive reward signal can be chosen. Exploration in a large discrete action space is a key challenge to scale RL to real-world problems, and remains an open research problem [273]. Existing RL-based language systems avoid this issue, by either including a pre-training Supervised Learning phase for the Language Model agent [262, 52], using an hybrid SL and RL objective [238], or only learning a simplified language restricted to a small vocabulary and very short sentences [296].

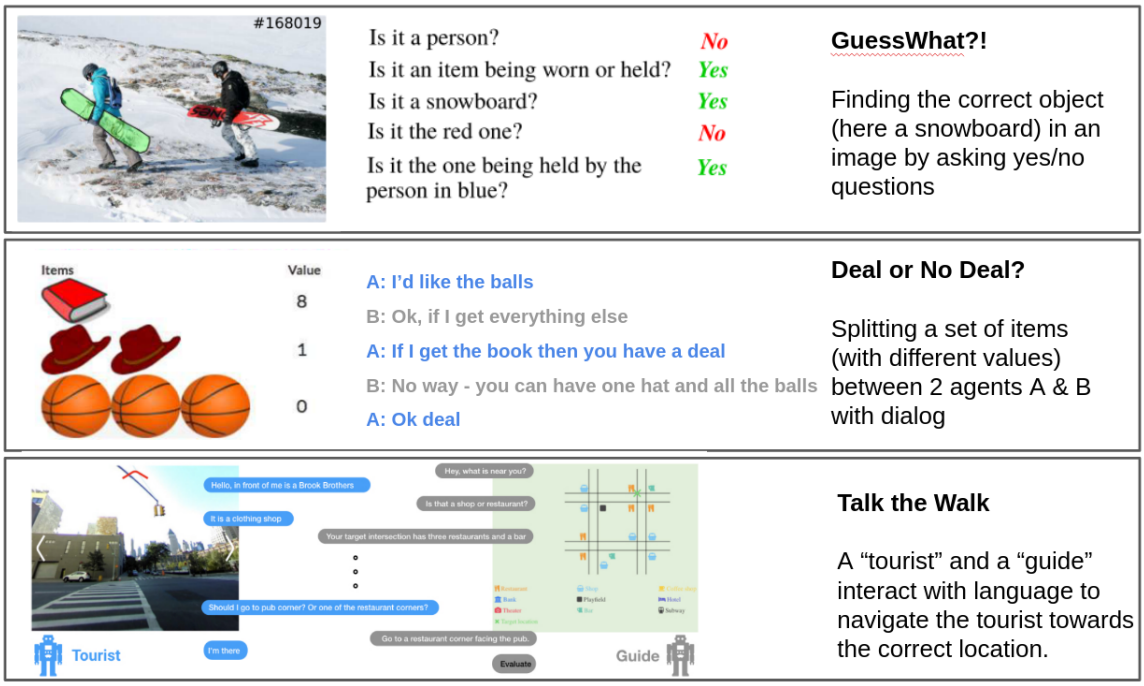


FIGURE 2.2 – Example of three Visual Dialog tasks that can be learned with Reinforcement Learning. GuessWhat ?! [55] is a dialog game to find the correct object in an Image with questions expecting a yes or no answer. Deal or No Deal ? [164] is a dialog negociation game to split a set of items (with different values) between two negociators. The Talk the Walk [56] creates a dialog game between a tourist and a guide based on a 2D grid environment of NYC neighbourhoods, to help the tourist navigate towards the right location.

2.2 Research Problems

In this thesis, we focus on improving Natural Language Generation systems, both on the model side, and on the learning side (i.e the way Language Models are trained). On the model side, we tackle the pitfalls of large-scale deterministic networks for sequential data through the lens of generative modelling. On the learning side, we tackle the pitfalls of the two modern approaches for training Language Models previously described. More precisely, we are interested in solving the following research questions.

1. How to plug Sequential Monte Carlo Methods into Deep Learning architectures, with the aim of designing Deep Generative Models well fitted for sequential data ?
2. How to design Sequential Monte Carlo algorithms to estimate state space models when either the dynamics of the latent state or the conditional likelihood of an observation given a state is intractable, which is the case for deep generative models ? How to do so while simultaneously reducing significantly the computational time of the existing numerical solutions ?
3. How to improve Natural Language Generation in SL-based Language Models, in particular foster diversity when generating text ?
4. How to solve the Action Space exploration challenge in RL-based Language Models, so that they can be

learned from scratch, while scaling to large vocabularies ?

Deep Generative Models for sequential data using SMC approaches. In the context of Language Modelling, we are interested in designing deep learning-based models that represent the stochastic nature of Language through (unobserved) latent states, as opposed to classic deterministic architectures simply using a softmax layer to model the LM probability distribution. Several attempts have been made to adapt existing DGM architectures to natural language generation problems, such as VAE [49, 76, 90, 167] or GAN [84, 251, 54]. However, the former suffer from the "posterior collapse" problem (which make them degenerate to deterministic networks), while the latter are notably ill-fitted for generating text at inference [30, 35]. Indeed, they showcase in particular some "mode collapse", i.e the model only learns a few modes of the distribution, outputting in practice very few different sentences. In this thesis, we focus instead on using Sequential Monte Carlo (SMC) methods to approximate the distribution of the latent states of a DGM given the observations (i.e the sequential input data), which allows to model complex latent distributions. SMC algorithms have been plugged to deep neural networks only in a handful of research works, and always within a Variational Inference framework [203, 191, 198, 189]. Our research works propose instead to directly estimate the true log-likelihood of the observations of DGMs through SMC approaches. We first develop a deep generative model based on the Transformer network, whose latent states and training objective are estimated with basic smoothing algorithms. Interestingly, our deep generative model framework goes beyond language modelling, and also allows to model uncertainty in sequential data problems. As a start, we thus showcase our model on uncertainty quantification tasks for time-series forecasting. Yet, scaling computationally costly SMC methods to deep neural networks and large-scale datasets is a research challenge by itself. Hence, in a second research work, we also focus on developing more efficient smoothing algorithms (a subset of SMC methods), whose reduced computational complexity make them more appealing for deep learning frameworks.

Truncated Reinforcement Learning for Natural Language Action Spaces. The fourth question has been barely explored by the RL and NLP research community. Dulac-Arnold et al. [73], Tennenholtz and Mannor [273], Chandak et al. [43] develop a generic RL algorithm for large discrete action spaces, that embed the actions into a continuous space : from there, classic RL algorithms for continuous control are used to learn a policy over the continuous space. Yet, these algorithms do not take in account the specificity of Natural Language, making them ill-fitted when scaling up to complex language tasks. Zahavy et al. [300], Seurin et al. [254] propose Q-learning algorithms with an elimination signal to eliminate forbidden actions, hence restricting the effective action space to a smaller set. Yet, eliminating actions from a set with several tens of thousands is still quite computationally expensive. Additionally, the use of Q-learning (ill-fitted on large action space) exacerbates the difficulty of the method. In this thesis, we chose to specifically leverage the structure of Language to truncate the Natural Language Action Space to a reasonable set of words (i.e less than 50 for large vocabularies). To model the structure of Language while remaining in an

end-to-end learning framework, we use the linguistic representation of a generic pretrained Language Model. From a linguistic point of view, this gives a generic linguistic prior knowledge to the computational Language Model, which then learns through RL the language pragmatics of the specific NLG task. From a model point of view, this bridges RL-based Language Models with pretrained language models in a novel way through a simple setting. In particular, such an approach avoids the numerical challenges arising when fine-tuning such pretrained language models with RL. Indeed, first, the latter approach particularly suffers from instability issues, given the difficulty of training Deep RL models on one side, and fine-tuning billions parameters networks on the other side [65, 160, 199, 233, 45]. Additionally, Transformers have empirically shown bad performances when trained by RL [215].

2.3 Thesis outline

Chapter 3 and 4 provide the background and the state-of-the-art relative to Language Models based on Deep Learning architectures. Chapters 4,5 and 6 present the research works developed within this thesis.

In Chapter 3, we describe the state-of-the-art deep learning models for sequential data. We start first with deep learning basics, then focus on neural network architectures for sequence modelling, from Recurrent Neural Networks to Transformers. Finally, we present Deep Generative Models for sequential data problems, and focus in the last section on Sequential Monte Carlo Methods, sampling-based algorithms that allow to approach the intractable score function of such Deep Generative Models.

In Chapter 4, we first describe in details Language Modelling and Natural Language Generation. We then present the theoretical framework for SL-based Language Models and RL-based Language Models. Finally, we describe the evaluation metrics that measure the quality and performance of Language Models.

Chapter 5 introduces TrufLL, for TRUncated ReinForcement Learning for Language. TrufLL develops a Reinforcement Learning framework for learning Language Models from scratch, by truncating the Vocabulary Action Space with a pretrained Language Model. It is applied on two Visual Question Generation (VQG) tasks : given an (image, answer) pair, the computational agent must generate a question on the image resulting in the correct answer. The chapter is completed by Appendix A, which gives more details about the hyper-parameters of the evaluated models, presents additional experiments, details the human evaluation study, and displays additional text samples generated by the evaluated models.

Chapter 6 introduces the Sequential Monte Carlo (SMC) Transformer, a Deep Generative model for sequential data. In a regression setting, the SMC Transformer allows to output a predictive distribution, instead of single-point estimates. In a classification setting, it models the variability of the input observations not simply through a softmax output, but also through stochastic latent states. We showcase first the SMC Transformer on multi-step time-series forecasting problems, and demonstrate its ability to quantify well the predictive uncertainty, unlike classical Bayesian Networks. Secondly, we apply it to Language Modelling tasks, where we demonstrate the remaining challenges to

scale such models on complex Natural Language Generation tasks. The chapter is completed by Appendix B with displays further details about the hyper-parameters of the evaluated models and the real-world time-series datasets, and presents additional results tables.

Chapter 7 introduces Backward Importance Sampling (BIS), a novel online smoothing algorithm to estimate the intractable log-likelihood of complex state-space models. BIS has much lesser computational cost than other state-of-the-art smoothing algorithms, while being applicable to a wider scope of state-space models. The algorithm is applied for state estimation and parameter estimation on several state-space models, in particular on a Recurrent Neural Network and in a multi-variate stochastic Lokta-Volterra Model, for which the density transition is intractable and is estimated using a pseudo-marginal framework. The chapter is completed by Appendix C, which details the framework to estimate the transition density of partially observed stochastic differential equations, by introducing General Poisson Estimators and Parametrix Estimators.

Finally, in chapter 8, we summarize the thesis research works and main contributions, and provide potential future research directions to go deeper into the research problems tackled within this thesis.

2.4 Contributions

The contributions of this thesis are listed hereafter.

— In chapter 5, we propose the first RL algorithm to train conditional Language Models from "scratch" (i.e without a SL pretraining phase), that scales to large vocabularies (around 15k words) and achieves performances close to RL-based Language Models with a pretraining phase.

  i) The algorithm proposes a novel way to combine RL-based Language Models with pretrained Language Models, through a truncation mechanism for the Natural Language Action Space, see Section 5.3.

  ii) The algorithm constitutes a first step towards getting rid of human-labelled datasets for learning conditional language models, while presenting an interesting language distribution, i.e. more original than models including a Maximum Likelihood Estimation phase, see Section 5.5.

— In chapter 6, we propose the SMC Transformer, a novel self-attention model combined with an algorithm using Sequential Monte Carlo Methods to directly estimate the score function of a Deep (stochastic) Transformer Neural Network through Fisher's identity, see Section 6.3 and Lemma 1.

  i) We demonstrate empirically that the model estimates correctly the variability of the observations for models with known observation noise, unlike common approaches used for uncertainty quantification, such as Bayesian Networks, see Section 6.4.2.

  ii) In a real-world time-series setting, we demonstrate empirically that the SMC Transformer outperforms classic approaches on common uncertainty quantification metrics, see Section 6.4.3.

— In chapter 7, we propose a novel smoothing algorithm for online estimation of state-space models (SSM) which relies on a Pseudo-marginal backward importance sampling step, see Section 7.4.

i) The algorithm has much less computational complexity than comparable smoothing algorithms, while exhibiting similar performances. It hence provides a first step towards scaling smoothing techniques for deep learning architectures, and we applied it for state estimation in a stochastic Recurrent Neural Network, see Section 7.5.1.

ii) The algorithm is also applicable on a wider scope of state-space models than previous approaches : its uses of pseudo-marginal techniques make it applicable on state-space models with intractable transition or observation density. We applied it first for online recursive maximum estimation on a Sine Model (Sections 7.5.2 and 7.5.3), for which we provide a novel estimator for the unknown transition density and its gradient based on a General Poisson Estimator, see Proposition 1 and Proposition 2 in Appendix C.1. Then, in the context of multi-dimensional diffusion process, we combine Parametrix Estimators [6] with the Wald's Trick [80] to obtain an almost surely positive unbiased estimate of the transition density, see Section 7.5.4 and Appendix C.1.2.

2.5 Publications

The publications presented in this thesis are listed hereafter :

— A. Martin, C. Ollion, F. Strub, S. L. Corff, and O. Pietquin. The Monte Carlo Transformer : a stochastic self-attention model for sequence prediction. arXiv preprint arXiv :2007.08620, 2020.

The code for the paper is available here : https://github.com/AMDonati/SMC-T-v2.

In revision for publication in IEEE Transactions on Signal Processing.

— A. Martin, M.-P. Etienne, P. Gloaguen, S. L. Corff, and J. Olsson. Backward importance sampling for online estimation of state space models. arXiv preprint arXiv :2002.05438, 2020.

Part of the code for the paper is available here : https://github.com/AMDonati/backward-IS.

In revision for publication in Journal of Computational and Graphical Statistics.

— A. Martin Donati, G. Quispe, C. Ollion, S. L. Corff, F. Strub, and O. Pietquin. Learning natural language generation from scratch. arXiv preprint arXiv :2109.09371, 2021.

The code for the paper is available here : https://github.com/AMDonati/RL-NLP.

In revision for publication in Annual Meeting of the Association for Computational Linguistics (ACL).

Chapitre 3

Deep Learning Models for Sequential Data

3.1 Deep Learning basics

Neural networks are widespread parametric functions typically used for regression and classification problems. In a supervised learning setting, consider $X \in \mathbb{R}^{d_x}$ and Y (a discrete or continuous random vector) two random variables defined on a measurable space $(\Omega, \mathcal{F})$. A neural network f_θ with parameter $\theta \in \mathbb{R}^d$ may be used to propose a model for the unknown conditional distribution of Y given X. In a classification setting with M classes, $Y \in \{1, \dots, M\}$ is discrete, and $f_\theta(X) = \{\mathbb{P}_\theta(Y = m|X)\}_{1 \leq m \leq M}$, where $\mathbb{P}_\theta(\cdot|X)$ is the proposed categorical conditional distribution. In a regression setting, $Y \in \mathbb{R}^m$ is a random variable lying in a continuous state. In that case, the conditional distribution of Y given X can be modelled for instance as a Gaussian distribution centered at the output of the neural network $f_\theta(X)$ and with an identity covariance matrix (or with a covariance matrix also depending on θ).

The term "network" comes from the fact that $x \mapsto f_\theta(x)$ is typically a composition of multiple different functions ; each of them is referred to as a *layer*. For instance, a L-layer neural network can be decomposed as $f_\theta = f^{(L)}_{\theta_{(L)}} \circ \dots \circ f^{(1)}_{\theta_{(1)}}$, where $\theta = \{\theta_{(1)}, \dots, \theta_{(L)}\}$. The number of composition functions used to defined f_θ (i.e the number of layers) is called the *depth* of the model. The final layer of the neural network is called the *output layer*, while the other layers are called *the hidden layers*. Each hidden layer of a neural network is typically vector-valued, and the dimensionality of the layer is given by its number of units.

There are different neural network architectures, i.e different possible families of functions for each layer. The most basic architecture forms a **feedforward neural network** (sometimes abbreviated FFNN) - and also called a multilayer perceptron (MLP). In such models, the function $f^{(\ell)}_{\theta^{(\ell)}}$ of the FFNN ℓ-th layer is a composition of *a linear projection* and *non-linear activation function* φ :

$$f^{(\ell)}_{\theta^{(\ell)}} : h^{(\ell-1)} \mapsto \varphi(W_\ell^T h^{(\ell-1)} + b_\ell), \tag{3.1}$$

where, $\theta_{(\ell)} = \{W_\ell, b_\ell\}$, W_ℓ and b_ℓ are respectively called the weight matrix and bias vector of the layer, while $h^{(\ell-1)}$ and $h^{(\ell)}$ are respectively the input and output of layer ℓ (and by convention $h^{(0)}$ is the input vector X). The most popular choice of activation functions are *rectified linear units* (ReLU) [204], and is defined by : $\varphi(z) = max(0, z)$. Many other activation functions have been used and designed, such as the logistic sigmoid and the hyperbolic tangent activation functions, see for instance chapitre II.6 of [107].

Training a Neural Network. Neural network are trained with gradient descent algorithms to update iteratively the unknown parameters. In a supervised learning setting, most neural networks are trained using *maximum likelihood* inference, i.e the cost function $J(\theta)$ to be minimized is simply the empirical expectation of the negative log-likelihood over the training dataset $\mathcal{D} = \{(X_i, Y_i)\}_{1 \leq i \leq n}$. For a classification problem, $J(\theta)$ is equivalent to the cross-entropy between the neural network output and the true class, and is written as :

$$J(\theta) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{m=1}^{M} \mathbb{1}_{Y_i = m} \log \mathbb{P}_\theta(Y_i = m | X_i) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{m=1}^{M} \mathbb{1}_{Y_i = m} \log f_\theta(X_i)_m, \tag{3.2}$$

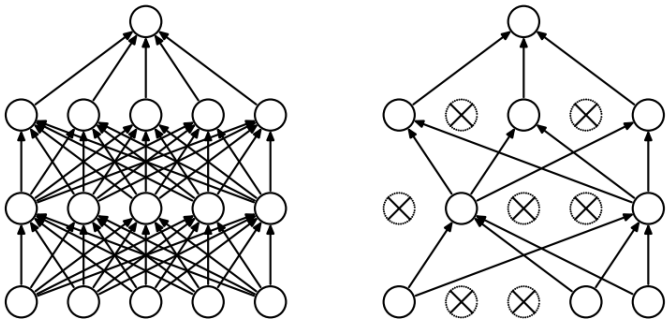
where $\{(X_i, Y_i)\}_{1 \leq i \leq n}$ are independent and identically distributed (i.i.d.) with same distribution as (X, Y), $\mathbb{1}$ designs the indicator function, and $f_\theta(X_i)_m$ is the m-th component of the vector $f_\theta(X_i)$. For a regression problem, $J(\theta)$ is the mean squared error between the neural network output and the true target, and is written as :

$$J(\theta) = -\frac{1}{2n} \sum_{i=1}^{n} \|Y_i - f_\theta(X_i)\|^2. \tag{3.3}$$

The gradient descent algorithm requires computing the gradient of $\theta \mapsto J(\theta)$. This can be very costly as it requires evaluating the model at every sample (X_i, Y_i) of the training dataset $\mathcal{D}$. In practice, we can compute this gradient by randomly drawing a small subset of samples from the training dataset (referred to as a *batch* $\mathcal{B}$ whose size is denoted by $|\mathcal{B}|$), and then by computing the average over these samples as an estimation for the expectation.

In Neural Network terminology, the process of outputting a prediction from an input X_i, $1 \leq i \leq n$, is referred to as *forward propagation* : information flows forward through the network. During training, forward propagation continues onward until it produces a scalar cost $J(\theta)$. On the other hand, the **backpropagation algorithm** [159] (often called only *backprop*) allows the information from the cost to flow backwards through the network, to compute the gradient of $J(\theta)$ with respect to the networks parameters θ. The backpropagation algorithm leverages the fact that a neural network is a composition of multiple functions, and relies on the chain rule, with a specific order of operations that is highly efficient.

Regularization techniques for Neural Networks. Neural Networks may depend on millions or billions of parameters, which makes them particularly prone to overfitting. The most popular deep learning regularization method is dropout [260], illustrated in Figure 3.1. For each layer $1 \leq \ell \leq L$, of the base network, each unit is retained randomly



(a) Standard Neural Net (b) After applying dropout.

FIGURE 3.1 – Graphical Illustration of dropout. (a) represents the base network while (b) represents a sub-network where random units have been dropped out. Source : [260].

with a fixed probability p_ℓ (p_ℓ is chosen by the user and is typically between 0.5 and 1), and a base-network is formed by all the retained units. In practice, a layer ℓ of a feedforward network with input $h^{(\ell-1)}$ using dropout samples a vector of independent Bernoulli random variables, each of which has a probability p_ℓ of being 1. The initial input $h^{(\ell-1)}$ is then multiplied element-wise by this vector to create a thinned output $\tilde{h}^{(\ell-1)}$. The only difference is that for each training sample of a mini-batch $\mathcal{B}$, we sample a sub-network by dropping out units, and forward and backward propagation are done only on this subnetwork. At inference, as it is not feasible to explicitly average the predictions from exponentially many sub-networks, an approximate average method is used, consisting in using a single neural network without dropout, whose weights $\tilde{W}_\ell$ are scaled-down versions of the trained weights $\tilde{W}_\ell = p_\ell W_\ell$.

In a Bayesian perspective, Monte Carlo dropout [94] considers that the parameters $\{W_\ell\}_{1\leq\ell\leq L}$ defined in (3.1) are assumed to be random and endowed with a prior distribution (typically a centered Gaussian distribution). The posterior distribution of $\{W_\ell\}_{1\leq\ell\leq L}$ given the dataset is intractable. Therefore, Monte Carlo dropout considers a variational inference framework where this posterior distribution is approximated by the best candidate in a parametric family of distributions. The candidate distributions are defined as follows : $\{W_\ell\}_{1\leq\ell\leq L}$ are independent and for all $1 \leq \ell \leq L$, the columns of W_ℓ are randomly set to zero using independent Bernoulli random variables with parameter $1 - p_\ell \in (0, 1)$. This allows to provide a variational distribution which is highly multi-modal. The model is then estimated by minimising a Monte Carlo estimator of the Kullback-Leibler divergence between the approximate posterior and the posterior of the full neural network. At inference, Monte Carlo dropout allows to output a predictive distribution whose samples come from a forward pass on the neural network with dropout activated. Such a technique is evaluated against our newly developed generative model in Chapitre 6.

Other regularization techniques include Parameter Norm Penalty ($\mathrm{L}^1/\mathrm{L}^2$ parameter Regularization), early stopping (going back to the parameter setting with the lowest validation error), dataset augmentation (increasing artificially the training dataset with fake examples) or for classification tasks, label smoothing [269] (hard 0 and 1 ground-truth labels are replaced with smoothed $\varepsilon/(M-1)$ and $1-\varepsilon$ versions for the M classes).

Optimization techniques for training Neural Networks. Optimizing complex non-convex functions containing from thousands through billions of parameters present numerous challenges and difficulties, such as getting stuck in local minima or saddle points. Optimising very Deep Neural Networks (and hence very deep computational graphs) results also in cliff regions and *exploding gradients* on one side, and *vanishing gradients* on the other side. Cliff regions and exploding gradients result from the multiplication of numerous large weights together (from each layer of the neural network). Such a phenomena can move suddenly the parameters extremely far in the parameter space (from one cliff structure to another), making learning unstable. One classic technique to overcome this challenge is *gradient clipping* [216], which clips the norm of the gradient so that it does not exceed a threshold value. On the other hand, vanishing gradients result from the multiplication of several very small weights together. Such a phenomena makes it difficult to know the direction the parameters should move to, in order to improve the cost function.

To overcome some of these difficulties, instead of using a simple stochastic gradient descent algorithm, sophisticated optimizers have been designed to better train neural networks. In this work, we will use the Adam optimizer [141], which accelerates training by including (i) a momentum variable giving a direction and speed at which the parameters move through the parameter space, and (ii) parameter-wise adaptive learning rates.

3.2 Network architectures for sequential data

In this section, we set the focus on neural networks that have been developed to process and model sequential data, such as time-series, text, speech, etc. We consider an input sequence of size T denoted by $X = (x_1, \dots, x_T)$. We will refer as o_t the logits output of the neural network for element x_t. In a regression setting, the logits output is directly the final output of the neural network. In a classification setting, the logits output is the pre-activation output of the last layer, i.e the output before applying a sigmoid function for a binary classification, or the output before applying a softmax function for a multi-class classification.

3.2.1 Recurrent Neural Networks

Vanilla Recurrent Neural Networks (RNNs). A recurrent neural network processes sequentially the input sequence and maintains at each time step t, a recurrent hidden state h_t that summarizes the past processed sequence $(x_1, x_2, ..., x_t)$. The recurrent dynamical system computes the hidden state h_t from the previous hidden state h_{t-1} and the current input element x_t. The output o_t of the network is then a function of the hidden state h_t :

$$\begin{aligned} \text{hidden layer :} \quad & h_t = a(Vh_{t-1} + Ux_t + b) \\ \text{output layer :} \quad & o_t = f(Wh_t + c)\,, \end{aligned} \tag{3.4}$$

where U, V, and W are weight matrices, while b and c are bias vectors, and a is an activation function (usually the $\tanh$ function). Multi-layers RNNs stack several hidden layers characterized by the first line of Equation 3.4. It thus maintains several hidden states h_t^ℓ taking as input the previous layer hidden state $h_t^{\ell-1}$ of $\ell > 1$, or the input data for the hidden state of the first layer. The output o_t is then based on the hidden state of the last layer h_t^L.

Vanilla RNNs suffer from the vanishing gradient problem mentioned in Section 3.1, and hence have a lot of difficulties to model long-term dependencies. Gated Recurrent Neural Networks, which maintain some memory states that help remembering the important information about the past inputs, have been thus developed and are now the most-widely used RNN variants for sequential data.

Gated Recurrent Neural Networks. The Long-Short Memory Network (LSTM) [125] maintains two recurrent states. The hidden state h_t plays the same role as the hidden state of vanilla RNNs, i.e it represents and summarizes the input sequence until timestep t. The cell state c_t represents a kind of additional memory, that keeps important information about the past for the current element being processed x_t, and discards irrelevant information. When processing input x_t, a LSTM has three gates, that control the information that flows into the cell state and the update of the hidden state. Their role and mathematical formulation are described in the next equations. Let $\tilde{h}_t = (h_{t-1}, x_t)$ be the vector obtained after concatenation of h_{t-1} and x_t.

Forget gate : controls the information discarded from the cell state.

$$f_t = \sigma(W_f \tilde{h}_t + b_f);$$

Input gate equation 1 : controls the new information stored in the cell state.

$$i_t = \sigma(W_i \tilde{h}_t + b_i);$$

Input gate equation 2 : new candidate values added to the cell state.

$$\tilde{C}_t = \tanh(W_C \tilde{h}_t + b_C);$$

cell update from forget and input gates :

$$c_t = f_t * c_{t-1} + i_t * \tilde{C}_t;$$

output gate : controls the information coming from the cell state to update the hidden state.

$$\bar{o}_t = \sigma(W_o \tilde{h}_t + b_o);$$

Hidden state update from cell state and output gate

$$h_t = \bar{o}_t * \tanh(c_t);$$

where W_f, W_i, W_C, W_o are weight matrices, while b_f, b_i, b_C, b_o are bias vectors, $*$ designs the dot-product, and σ is the logistic sigmoid function. Figure 3.2 compares the recurrent mechanism in a vanilla RNN (a) and a LSTM (b). Another popular Gated RNN is the GRU [46] : containing only two input gates and not maintaining an additional

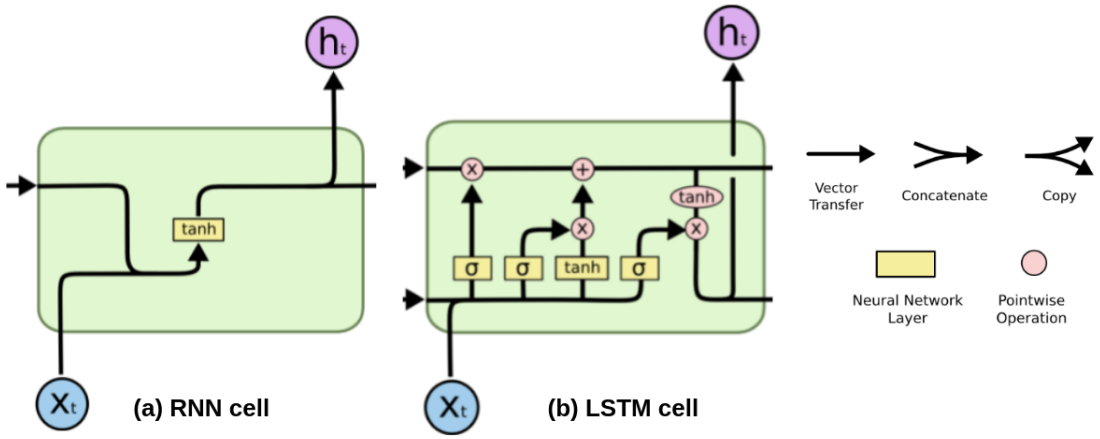


FIGURE 3.2 – Illustration of the recurrent modules (also called cell) that computes the new hidden state h_t from the previous one h_{t-1} and the current input x_{t-1} for (a) a vanilla RNN and (b) a LSTM. The legend illustrates the different operations taking place in the recurrent cell. The block in yellow represent a layer of a neural network with their corresponding activation function (tanh or sigmoid). source : https://colah.github.io/posts/2015-08-Understanding-LSTMs/

memory unit, it has lesser complexity, and has displayed similar empirical performances than the LSTM.

RNN encoder-decoder frameworks. Sequence to sequence models take as input the sequence $X = (x_1, ...x_T)$, for instance a sequence of words in a NLP setting, and output another sequence $Y = (y_1, ...y_{T_y})$, for instance another sequence of words. They can be built using a RNN encoder-decoder framework. For instance, in Neural Machine Translation, the input words sequence is a sentence in a source language, and the target words sequence is a sentence in the target language.

The encoder reads and encodes the input sentence $X = (x_1, ...x_T)$ into a vector c. The encoder f^{θ}_{enc} is a RNN variant (a vanilla RNN, a LSTM, a GRU, etc), and c is a function of the concatenation of each hidden state h_t :

$$\text{for all } t \in \{1, ...T\}, \quad h_t = f^{\theta}_{\text{enc}}(x_t, h_{t-1})$$
$$c = g^{\theta}_{\text{enc}}(\{h_1, ...h_T\})$$

A simple choice of function g^{θ}_{enc} is the last hidden state h_T that summarizes the input sequence : $g^{\theta}_{\text{enc}}(\{h_1, ...h_T\}) = h_T$.

The decoder is trained to predict the next word y_t given the context vector c and all the previous predicted words $\{y_1, ..., y_{t-1}\}$. It is also a RNN variant taking as input the decoded target sequence and the context c :

$$p_\theta(y_1, \ldots, y_{T_y}|x_1, \ldots, x_T) = \prod_{t=1}^{T_y} p_\theta(y_t|\{y_1, \ldots, y_{t-1}\}, c); \quad \text{where} \quad p_\theta(y_t|\{y_1, \ldots, y_{t-1}\}, c) = g^{\theta}_{\text{dec}}(y_{t-1}, s_t, c)\,,$$

where s_t is the hidden state of the decoder RNN, and g^{θ}_{dec} represents typically a feedforward neural network with a

softmax activation.

Adding attention to an encoder-decoder framework. The first attention mechanism was originally developed in [11] to better model alignment between the input and output sequences in Machine Translation. It has been extended since then to other kinds of alignment, such as Visual Attention between images and text [293]. Within an encoder-decoder framework, instead of considering a single context vector c, an attention mechanism considers a time-dependent context c_t. The time-dependent context c_t is called the *attention vector*, and is computed as follows. The attention model first maps the input sequence $X = (x_1, ...x_T)$ and a target word y_t to a set of *key-values* pairs (K, V) and a *query* q_t using parametric functions. The key-value pairs (K, V) represent the input sequence, while the query q_t represents the current decoded word y_t that attends to the input. The keys emphasize the elements of the input sequence that are relevant to the query. The values are additional representations of the input sequence, used to compute the weighted sum that creates the attention vector :

$$c_t = \text{att}(q_t, K, V) = \sum_{s=1}^{T} p(a(k_s, q_t))v_s, \tag{3.5}$$

where a is called the alignment function and p the distribution function (typically the softmax, but other distributions have been considered such as logistic sigmoid or sparsemax [193]). They determine how keys and query are combined to produce attention weights. Figure 3.3 illustrates a RNN encoder-decoder framework with (subfigure (b)) and without attention (subfigure (a)).

3.2.2 Transformers networks

The Transformer is a novel neural network architecture for sequence modelling proposed in 2017 in the paper "Attention is all you need" [277]. As it is hinted in the title, the Transformer is a encoder-decoder framework entirely relying on a particular kind of attention - called "self-attention" - to model dependencies in the input or output sequences. Originally developed to address the high computational cost characterizing the training of RNNs, it eschews recurrence, allowing thus to process input sequences all at once, instead of element per element.

Figure 3.4 displays the Transformer architecture. The network includes the following components :

1. **Self-attention** layers (in orange), the core operation that models dependencies in the input or output sequences. "Multi-Head" refers to several self-attention functions computed in parallel.
2. **Positional encodings** that encode the position of each element in the input sequence. As there is no recurrent mechanism processing the sequence sequentially, they allow to model the order of elements in the input sequence.
3. Additional **residual layers** that transform the output of the self-attention (in yellow and blue).

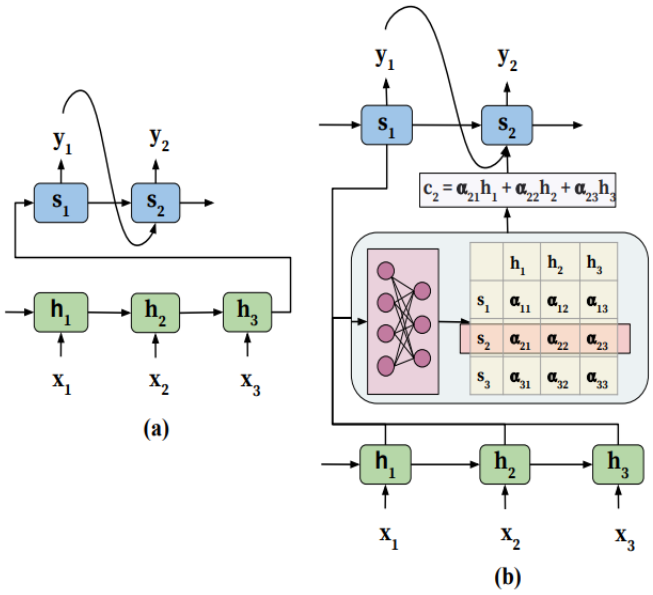


FIGURE 3.3 – Illustration of sequence to sequence model with (figure (b)) or without (figure (a)) attention. In (a), the current word being decoded y_t depends on the input sequence X only through the initialization of the initial state of the decoder, which is equal to the encoder final hidden state. In (b), y_t depends on a attention vector c_t (itself a function of the input sequence X with learned attention weights) through the current decoder hidden state s_t used to decode y_t. Source : [44].

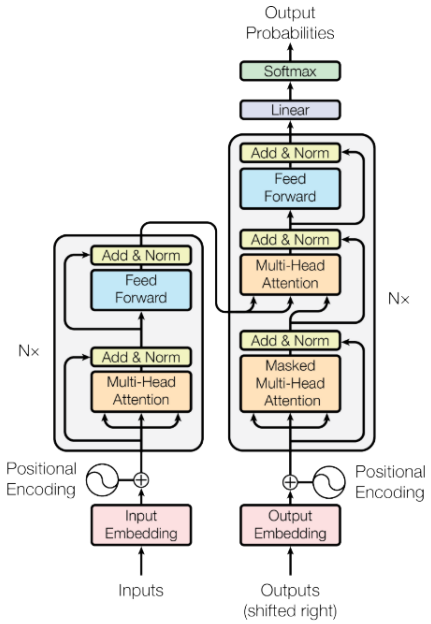


FIGURE 3.4 – The Transformer architecture. Source : [277].

Self-attention is an attention model, that maps an input sequence to queries, and a set of key-values pairs. The query represents the element x_t of input sequence $X = (x_1, ..., x_T) \in \mathbb{R}^{T \text{ x } d_x}$, and the set of keys-values pairs are used to compute the attention vector z_t for input x_t. The set of queries $Q = (q_1, ..., q_T) \in \mathbb{R}^{T \text{ x } d_k}$ and the set of

keys-values pairs $\left(K = (k_1, ..., k_T) \in \mathbb{R}^{T \text{ x } d_k}, V = (v_1, ..., v_T) \in \mathbb{R}^{T \text{ x } d_v}\right)$ are computed as linear transformations of X :

$$Q = XW^Q; \quad K = XW^K; \quad V = XW^V; \tag{3.6}$$

where W^Q, W^K, and W^V are weight matrices respectively in $\mathbb{R}^{d_x \text{ x } d_k}$, $\mathbb{R}^{d_x \text{ x } d_k}$, and $\mathbb{R}^{d_x \text{ x } d_v}$, with d_k the dimension of each query and key, d_v the dimension of the each value, and d_x the dimension of the embedding of each element of X. The attention tensor $Z = (z_1, ..., z_T)$, where $z_t \in \mathbb{R}^{d_v}$, that computes the attention vector for each element of X is :

$$Z = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{3.7}$$

where $\text{softmax}(QK^T/\sqrt{d_k})$ are the attention weights of the attention model. Self-attention is sometimes referred to as "scaled dot-product attention", "scaled" coming from the division by $\sqrt{d_k}$ and "dot-product" from the fact that the attention weights are computed with the dot-product of a query with all keys.

Multi-head attention. Instead of performing a single self-attention operation, the self-attention layer performs H attentions in parallel, with different learned projections weights $(Q^h, K^h, V^h) = (XW_h^Q, XW_h^K, XW_h^V)_{h=1}^H$. Each attention vector $\text{Attention}(Q^h, K^h, V^h)$ is referred to as an head. The final attention tensor Z is formed by a projection of the concatenation of all attention heads, as detailed below.

$$\text{for all } h \in \{1, ..., H\}, \quad \text{head}_h = \text{Attention}(Q^h, K^h, V^h) \tag{3.8}$$

$$Z = \text{MultiHead}(Q, K, V) = [\text{head}_1, ..., \text{head}_H]W^O \tag{3.9}$$

where W^O is a learnable matrix weight of size d_{model} x d_{model}, with $d_{\text{model}} = H * d_v$.

Multi-Head self-attention allows the Transformer to jointly attend to information from different subspaces at different positions. Thus, an "head" of attention can be understood as a specific kind of attention that models the dependencies in the input sequence. For instance, in a NLP setting, we might want to have "semantic" attention that models semantic dependencies between words, as well as "syntactic" attention, that models syntactic dependencies between words. Figure 3.5 shows an example of attention weights per head for the word "it" of the input sentence "The animal didn't cross the street because *it* was too tired".

In the Transformer architecture displayed in Figure 3.4, we see that multi-head self-attention is used in three ways in an encoder-decoder framework. As the mechanism that models dependencies in the sequential data being processed (input sequence) and the one being decoded (output sequence), the self-attention block of the encoder and the first one of the decoder represents hidden representations of such sequence. The attention vector Z thus plays the same role as the sequence of hidden states in a recurrent neural network architecture. In the decoder, a mask over self-attention is needed to represent the output sequence : the masks zero out the future time steps for

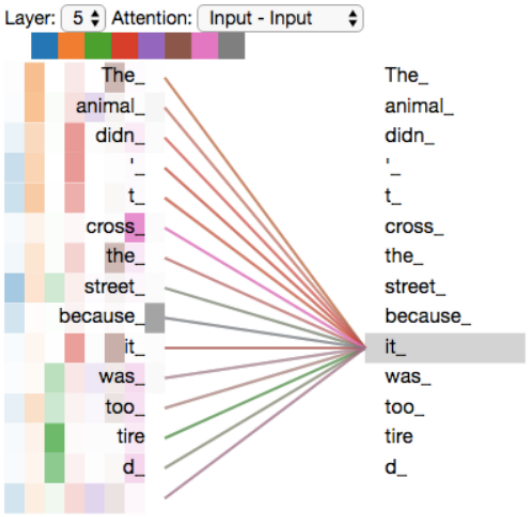


FIGURE 3.5 – Visualization of the attention per head in a self-attention layer of a Transformer, for the input sentence "The animal didn't cross the street because it was too tired." Source : https://jalammar.github.io/illustrated-transformer/

a given word being decoded, to model the auto-regressive character of the LM (see Section 4.1 for more details on LM). Finally, the second self-attention block of the decoder takes as input the encoder output (to compute the set of key-values pairs) and the output of the decoder first self-attention block (to compute the query). It thus models a classic attention mechanism in a encoder-decoder framework to model dependencies between the input sequence and target sequence.

Residual layers. For each self-attention block, the attention vector is further transformed first by a residual connection [121] followed by a layer normalisation [10] ("Add & Norm" block of Figure 3.4), secondly by a pointwise feed-forward neural network (two linear transformations with a ReLU activation in-between), and finally by a second "Add & Norm" block. Two dropout layers are inserted within the residual layers, the first one is applied directly to the attention vector, and the second one at the output of the pointwise feedforward neural network, before the last "Add & Norm" Block. Additionally, another dropout layer is applied at the encoder input, after the embedding layer, and before the first self-attention block. The original Transformer was trained with a dropout rate equal to 0.1. As illustrated in Figure 3.4), the original Transformer architecture stacks several blocks of self-attention and residual layers in the encoder and decoder. The logits output of the network that models $p_\theta(y_t|\{y_1, ..., y_{t-1}\}, X)$ (where X is the input sequence and $Y = (y_1, ...y_{T_y})$ is the output sequence) is obtained by a last linear transformation of the output of the last block. In a NLP setting, this projection is followed by a softmax activation, so that $p_\theta(y_t|\{y_1, ..., y_{t-1}\}, X)$ is a categorical distribution over a vocabulary $\mathcal{V}$.

Although the original Transformer has been developed as an encoder-decoder framework to perform sequence to sequence learning tasks such as Machine Translation, taking only the Transformer decoder provides an ar-

chitecture for Language Models, and plenty of further works on Language Models have only used the decoder architecture [230, 232, 33].

3.3 Deep Generative Models for sequential data

The neural network architectures described in the previous section have been specially designed to model complex long-range dependencies within sequential data. Over the last decade, they have achieved impressive performances in a wide variety of sequential modelling tasks, from multivariate time-series forecasting [78, 171, 259], through speech recognition to complex Natural Language Understanding (NLU) [221, 65, 183] and Natural Language Generation (NLG) tasks [50, 230, 232, 33]. Yet, they are based on a deterministic modelling of a supervised dataset. Indeed, in a regression setting (e.g when modelling time-series for example), these networks only output a single estimate of a time-series. In a classification setting, although the softmax output provides a probability distribution over the M classes, it is the only source of stochasticity, and fails in practice to model the variability of the observations, and to quantify the uncertainty of the neural predictor in its predictions (section 5.1 of [93]).

On the other hand, state-space models (part I of [40]) allow to represent the temporal evolution of complex non-linear dynamic systems in a probabilistic framework. They are composed of latent states $(X_t)_{0\leq t\leq T}$, partially observed through a noisy sequence of observations $(Y_t)_{0\leq t\leq T}$. In standard *Hidden Markov Models* (HMM) [229], $(X_t)_{0\leq t\leq T}$ is assumed to be a Markov chain. For each $0 \leq t \leq T$, the conditional distribution of X_t given $X_{0:t-1}$ depends on X_{t-1} only. On the other hand, the observations $(Y_t)_{0\leq t\leq T}$ are assumed to be independent given $(X_t)_{0\leq t\leq T}$ and the conditional law of Y_t depends on X_t only. The dependency structure of a Hidden Markov Model is illustrated in Figure 3.6.

Deep Generative Models (DGM) bridge Neural Networks with state-space modelling. Based on neural architectures, they view any kind of observed dataset as a finite set of samples from an underlying distribution. In the context of supervised learning, instead of outputting only single-point estimates of a given target variable Y, they output a predictive distribution. For sequential data, in a time-series forecasting setting, such models are hence able to output predictive intervals, and thus provide an uncertainty estimate of the forecast. In a language modelling setting, such models would be able to better model the diversity naturally found in natural language (i.e there is various ways to express a same idea/meaning).

Various kinds of Deep Generative Models have been developed, with (i) different ways of modelling stochasticity within the neural network, (ii) different methods for estimating the (usually) intractable training objective and hence to train the DGM, and (iii) various neural network architectures that form the basis of the DGM. Modern DGM include Variational Auto-Encoders (VAE) [143], Generative Adversarial Networks (GAN) [106], Normalizing Flows [66, 142], and Energy-Based Models [112, 129]. In the next section, we will focus on DGM that uses Sequential Monte-Carlo (SMC) Methods to estimate the log-likelihood of the observations. For further details on DGM, we refer to [29]

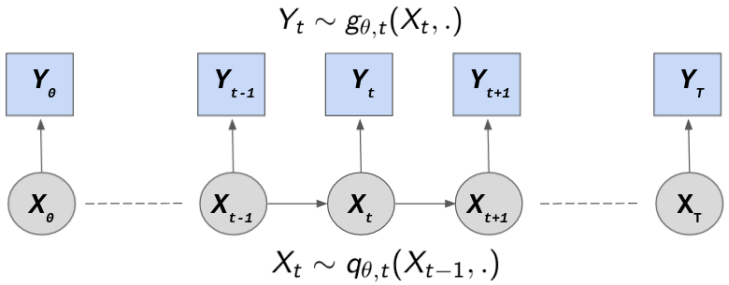


FIGURE 3.6 – Illustration of the dependency structure of a Hidden Markov Model (HMM). Latent states are circled in grey, and observations are squared in blue. The current state X_t depends only on the previous state X_{t-1} through probability density function $q_{\theta,t}$, while the current observation Y_t depends only on the current state X_t through probability density function $g_{\theta,t}$.

(modern approaches for DGM) and [140] (focus on DGM for Natural Language).

Limits of VAE-based approaches for sequential data. Easy to implement and computationally efficient, VAE are popular deep generative models. Initially applied to Computer Vision problems, multiple research works have extended them to sequential data problems [49, 90, 76, 203, 191, 63, 167]. However, such models suffer from several pitfalls. First, the classic VAE considers a Gaussian distribution for the posterior variational distribution : this conveniently allows to compute an exact analytical expression for the ELBO KL divergence term and to use the reparametrization trick [144], which makes the sampling of the latent variable differentiable. Yet, in practice, this outputs a predictive distribution known to be ill-fitted for estimating certain families of distributions, for instance multimodal distributions. Secondly, combining a VAE with recurrent encoder-decoder framework to model sequential data leads to the notorious KL vanishing issue (also referred to as the posterior collapse) [30, 288] : the encoder produces posteriors almost identical to Gaussian prior for all sentences, and the decoder completely ignores the latent variable, hence reducing to a deterministic decoder. Although several numerical tricks have been developed to mitigate this issue [166, 120], such models are thus harder to train, and having a meaningful latent space for VAE-based RNN or Transformers remains an open problem.

In the next section, we introduce Sequential Monte Carlo (SMC) methods, sampling-based algorithms to learn generative models for sequential data with latent variables, that could help learn richer latent variable distributions than VAEs. In Chapter 6, we introduce the Sequential Monte Carlo Transformer, a novel Transformer-based generative model for sequence modelling, which is trained using SMC methods.

3.3.1 Sequential Monte Carlo Methods for latent-variable models

This section details the framework to introduce Sequential Monte Carlo Methods, a set of sampling-based algorithms to estimate the (intractable) distribution of latent states given the sequence of observations in state-space models. Such algorithms are used and developed in chapters 6 and 7. Please notice that this section introduces numerous equations and mathematical notations ; the latter are summarized in the list of Symbols. To put in context the use of SMC approaches, we start with a concrete use-case representing a stochastic RNN as a state-space model. For the reader familiar with the subject, this section can be scanned quickly, or even skipped.

We consider a state-space model with parameter $\theta \in \Theta \subset \mathbb{R}^p$ ($p \geq 1$), characterized by a latent $\mathcal{X}$-valued Markov process $(X_t)_{0\leq t\leq T}$, and $\mathcal{Y}$-valued observations $(Y_t)_{0\leq t\leq T}$, with $\mathcal{X}$ and with $\mathcal{Y}$ two countably generated σ-fields. The state process has an initial probability density function χ_θ with respect to a given reference measure ν on $(\mathsf{X}, \mathcal{X})$, and a Markov transition kernel $Q_{\theta,t}$ on $\mathsf{X}\times\mathcal{X}$ with probability density $q_{\theta,t}(x,\cdot)$ with respect to ν. In this section, we will place ourselves in the context of Hidden Markov Models (HMM). In this setting, the set of observations $(Y_t)_{0\leq t\leq T}$ are independent conditionally on $(X_t)_{t\geq 0}$, and the conditional law of Y_t depends on X_t only, and has probability density function $y \mapsto g_{\theta,\ell}(X_t, y)$ with respect to a given reference measure λ on $(\mathsf{Y}, \mathcal{Y})$.

The following stochastic recurrent neural network (which only has hidden-to-hidden recurrent connections) would be a concrete example of a HMM. The hidden state is initialized as $X_0 \sim \mathcal{N}(0, \Sigma)$ and for all $t \geqslant 1$, the unobserved stochastic hidden state X_t and the next observation Y_t are computed as follows :

$$X_t = \tanh(W_1 X_{t-1} + b + \eta_t) \quad \text{and} \quad Y_t = g(W_2 X_t + c + \varepsilon_t),$$

W_1, W_2, and b, c are respectively the weight matrices and bias, and form the parameters θ of the state-space model. Σ is an unknown covariance matrix, $(\eta_t)_{t\geqslant 1}$ and $(\varepsilon_t)_{t\geqslant 1}$ are independent Gaussian random variables with covariance matrices Q and R. g is an activation function, typically a softmax function if the observations take discrete values, or the identity function if they are continuous vectors. This example is a simplified version of the model used in Section 7.5.1 to assess the performance of our proposed Backward Importance Sampler.

When training such a stochastic network, we are interested in inferring the parameters θ of the model that best predict the sequence of observations. The fundamental ingredient of Bayesian and maximum likelihood inference of the parameter $\theta \in \Theta$ is the likelihood function defined, for any sequence of observations $Y_{0:T}$, as :

$$L_{T,\theta} = \int \chi_\theta(x_0) g_{\theta,0}(x_0, y_0) \prod_{u=1}^{T} q_{\theta,u}(x_{u-1}, x_u) g_{\theta,u}(x_u, y_u) \nu(\mathrm{d}x_{0:T}). \tag{3.10}$$

In this framework, the likelihood function appears naturally as the normalizing constant of the posterior distribution of some hidden states given the observations : fixed interval smoothing distributions are defined, for any bounded measurable function h on $\mathsf{X}^{t_2-t_1+1}$, $\theta \in \Theta$, and any $0 \leq t_1 \leq t_2 \leq T$, by :

$$\phi^{\theta}_{t_1:t_2|T}[h] = [L_{T,\theta}]^{-1} \int \chi_\theta(x_0) g_{\theta,0}(x_0, y_0) \prod_{u=1}^{T} q_{\theta,u}(x_{u-1}, x_u) g_{\theta,u}(x_u, y_u) h(x_{t_1:t_2}) \nu(\mathrm{d}x_{0:T}). \tag{3.11}$$

In the case of a Hidden Markov Model with observations $Y_{0:T}$, this quantity may be interpreted as :

$$\phi^{\theta}_{t_1:t_2|T}[h] = \mathbb{E}_\theta \left[h(X_{t_1:t_2}) | Y_{0:T} \right],$$

where $\mathbb{E}_\theta$ is the expectation under the law of the model parameterized by θ. Maximum likelihood parameter estimates of θ may be then obtained for instance by applying the EM algorithm introduced in [62] or gradient ascent algorithms. For the latter algorithms, the associated score function is the log-likelihood of the observations :

$$\ell_T : \theta \mapsto \log L_{T,\theta} = \log \left(\int \chi_\theta(x_0) g_{\theta,0}(x_0, y_0) \prod_{u=1}^{T} q_{\theta,u}(x_{u-1}, x_u) g_{\theta,u}(x_u, y_u) \nu(\mathrm{d}x_{0:n}) \right). \tag{3.12}$$

Under regularity assumptions, Fisher's identity provides a relationship between the gradient of the log-likelihood score (the quantity of interest when training for instance the stochastic RNN with gradient descent) and the joint smoothing distribution $\phi^{\theta}_{0:T|T}$:

$$\nabla_\theta \ell_T(\theta) = \phi^{\theta}_{0:T|T} \left[\nabla_\theta \log p_\theta(\cdot, Y_{0:T}) \right],$$

where

$$p_\theta(x_{0:T}, Y_{0:T}) = \chi_\theta(x_0) g_{\theta,0}(x_0, Y_0) \prod_{u=1}^{T} q_{\theta,u}(x_{u-1}, x_u) g_{\theta,u}(x_u, Y_u).$$

Nevertheless, the conditional distribution $\phi^{\theta}_{0:T|T}$ is not available explicitly in nonlinear and non Gaussian state space models. For instance, in the stochastic RNN setting, this conditional distribution is intractable due to the non-linearity of activation functions. A solution to maximize the log-likelihood (3.12) is to obtain explicitly low variance Monte Carlo estimates of these smoothed expectations. Sequential Monte Carlo (SMC) methods [40] are a general class of Monte Carlo methods to estimate the likelihood of the observations and smoothing expectations of the form $\mathbb{E}_\theta \left[h_{0:T}(X_{0:T}) | Y_{0:T} \right]$), with $h_{0:T}$ a measurable function defined on X^{T+1}.

In SMC methods, we distinct two class of algorithms.

1. **Particle Filtering :** The filtering distribution at time t is the distribution of the state X_t of a state-space model, given all the observations $Y_{0:t}$ received up to time t. The Filtering problem hence aims at providing a sequential Monte Carlo approximation of the marginal distributions $\{\phi^{\theta}_{t|t}\}_{t \geq 0}$

2. **Particle Smoothing :** The joint smoothing distribution $\phi^{\theta}_{0:T|T}$ is the distribution of the state $X_{0:T}$ given all

the observations up to timestep T. Particle smoothing algorithms are computationally more challenging. In practice, efficient smoothing algorithms rely on numerical approximations that are based on Monte Carlo approximations of the filtering distributions.

In the next paragraphs, for all $1 \leq t \leq T$, we will note $\phi^{\theta}_{0:t|t}(x_{0:t})$ the joint smoothing distribution : the dependency on the observations $Y_{0:t}$ is kept implicit for simplifying notations.

Particle Filtering. Particle Filtering (PF) algorithms (also referred to as particle filters) are based on the following decomposition of $\phi^{\theta}_{0:t|t}$:

$$\phi^{\theta}_{0:t|t}(x_{0:t}) = \frac{g_{\theta,t}(x_t, Y_t)\phi^{\theta}_{0:t|t-1}(x_{0:t})}{\ell^{\theta}_{t|0:t-1}(Y_t, Y_{0:t-1})}, \tag{3.13}$$

where $\ell^{\theta}_{t|0:t-1}(Y_t, Y_{0:t-1})$ is the predictive distribution of Y_t given the past observations $Y_{0:t-1}$. Using that

$$\phi^{\theta}_{0:t|t-1}(x_{0:t}) = q_{\theta,t}(x_{t-1}, x_t)\phi^{\theta}_{0:t-1|t-1}(x_{0:t-1}) \tag{3.14}$$

we obtain

$$\phi^{\theta}_{0:t|t}(x_{0:t}) = \frac{g_{\theta,t}(x_t, Y_t)q_{\theta,t}(x_{t-1}, x_t)\phi^{\theta}_{0:t-1|t-1}(x_{0:t-1})}{\ell^{\theta}_{t|0:t-1}(Y_{0:t-1}, Y_t)}. \tag{3.15}$$

This gives a recursion formula between $\phi^{\theta}_{0:t-1|t-1}$ and $\phi^{\theta}_{0:t|t}$. Yet, while we would like to sample from $\phi^{\theta}_{0:t|t}$, in a general state-space modelling setting, it is usually impossible, and we use instead a sequential version of a sampling importance resampling (SIR) procedure [240, 241], by sampling from a proposal distribution $p^{\theta}_{0:t|t} = p^{\theta}_t$, and performing resampling with the importance weights.

Following this procedure, the standard particle filtering algorithm approximates the marginal filtering distribution at time t with a set of N weighted particle samples $\{(\omega^i_t, \xi^i_t)\}_{i=1}^N$. Samples $\{\xi^i_t\}_{i=1}^N$ are called particles, and $\{\omega^i_t\}_{i=1}^N$ are called resampling weights, or filtering weights. At time step t, $\{(\omega^i_{t-1}, \xi^i_{t-1})\}_{i=1}^N$ is transformed into a new weighted set of particle samples $\{(\omega^i_t, \xi^i_t)\}_{i=1}^N$, with two steps.

1. The particle selection step resamples N ancestor particles $(\tilde{\xi}^i_{t-1}))_{i=1}^N$ from $\{\xi^i_{t-1}\}_{i=1}^N$ using the previous resampling weights $\{\omega^i_{t-1}\}_{i=1}^N$.
2. The propagation (or mutation) step samples new particles $\{\xi^i_t\}_{i=1}^N$ using a conditional proposal distribution $p^{\theta}_t(\tilde{\xi}^i_{t-1}, .)$ (again, the dependency on Y_t is kept implicit), and then updates recursively the sampling weights $\{\omega^i_t\}_{i=1}^N$.

The complete algorithm is detailed in Figure 1.

The resampling step is crucial in Sequential Monte Carlo Methods, to limit the variance of the estimator of the filtering or smoothing distribution. Resampling discards particle with low-weights with a high-probability (that we do not want to carry forward), and allows to focus on regions on high-probability mass of the state space. Other

Algorithm 1 Standard Particle Filter.

INPUT : number of particles N, set of observations $(Y_t)_{1\leq t\leq T}$, proposal distribution p_t^θ, initial distribution for the initial state χ_0.
{Initialization} :
for $i = 1$ **to** N **do**
Sample Initial particles $\xi_0^i \sim p_0^\theta(X_0)$
Compute initial importance weights :

$$\omega_0^i = \frac{g_{0,\theta}(\xi_0^i, Y_0)\chi_0(\xi_0^i)}{p_0^\theta(\xi_0^i)}$$

end for
for $t = 0$ **to** T **do**
{Particle Selection} : Sample N indices $I_i \in \{1, ..., N\}$ according to the multinomial distribution from the previous resampling weights $\{\omega_{t-1}^i\}_{i=1}^N$
Set $\tilde{\xi}_{t-1}^i = \xi_{t-1}^{I_t^i}$ and $\tilde{\omega}_{t-1}^i = \frac{1}{N}$
{Particle Propagation} : Sample next particle $\xi_t^i \sim p_t^\theta(\tilde{\xi}_{t-1}^i, \xi_t^i)$
{Resampling weights update} :

$$\omega_t^i = \frac{g_{t,\theta}(\xi_t^i, Y_t)q_{t,\theta}(\tilde{\xi}_{t-1}^i, \xi_t^i)}{p_t^\theta(\tilde{\xi}_{t-1}^i, \xi_t^i)}$$

Normalize new resampling weights :

$$\omega_t^i \leftarrow \frac{\omega_t^i}{\sum_{j=1}^N \omega_t^j}$$

end for

unbiased resampling schemes than have been proposed in the litterature ; the most popular and efficient schemes besides multinomial sampling are systematic resampling and residual resampling [41].

The bootstrap filter [110] uses directly as proposal distribution the transition density $q_{\theta,t}$ to sample at time step t the next set of particles $(\xi_t^i)_{i=1}^N$. The computation of the current resampling weights hence simplifies to : $\omega_t^i = g_{t,\theta}(\xi_t^i, Y_t)$. In that case, the incremental weight does not depend on the past trajectory of the particle but only on the observation density $g_{t,\theta}(\xi_t^i, Y_t)$. This is the particle filter used for the SMC Transformer, described in Chapter 6.

Particle Smoothing. Various particle smoothing algorithms (also referred to as *smoothers*) relying on a particle filtering phase have been proposed in the SMC literature. We describe the most widespread, starting from the most basic one, the poor man smoother up to smoothers relying on the forward-backward decomposition of the smoothing distribution.

The **poor man smoother** (PMS) introduced in [145] approximates the joint smoothing distributions $\phi_{0:T|T}^\theta$ using the genealogy of the particles produced by a particle filtering algorithm. The genealogical trajectories are defined recursively and updated at each time step using the new particles and indices $\{(I_t^i, \xi_t^i)\}_{i=1}^N$. For all $1 \leq t \leq T$, for all $1 \leq i \leq N$:

$$\xi_{0:t}^i = (\xi_{0:t-1}^{I_t^i}, \xi_t^i) \tag{3.16}$$

where I_t^i is defined in algorithm 1. A last time step T, the joint smoothing distribution $\phi^\theta_{0:T|T}[h]$ is approximated, for any bounded and measurable function h on X^{T+1} by $\sum_{i=1}^T \omega_T^i h(\xi^i_{0:T})$. While the resampling step is crucial to ensure a low-variance of the particle filtering estimator, particle filters tend to suffer **from the path degeneracy issue**, as observed in [145, 146, 82, 226]. At each time step t, the resampling step selects $M \leq N$ past trajectories among the N possibles ones. As the number of resampling steps increases, the number of trajectories discarded increases, leading to a decreasing number of unique past genealogy trajectories for the current particle ξ_t^i. In particular, when considering full trajectories $\{\xi^i_{0:T}\}_{i=1}^N$, they are prone to share the same common ancestral path until some time step, meaning that the marginal distribution $\phi^\theta_{t|T}$ is approximated by very few different particles for $t \ll T$. One solution to overcome this phenomena consists in not resampling at every time step, but only when a specific criterion on the resampling weights is met. In practice, a popular solution resamples only when the variance of the unnormalized weights is superior to a specified threshold. The heuristics for the threshold is generally the Effective Sample Size (ESS) criterion, defined as follows : $\mathrm{ESS} = (\sum_{i=1}^N (\omega_t^i)^2)^{-1}$.

Smoother based on forward-backward recursions. The joint smoothing distribution $\phi^\theta_{0:T|T}$ may be factorized as follows, see for instance [212] and the graphical model given in Figure 3.6,

$$\phi^\theta_{0:T|T}(x_{0:T}) = \phi^\theta_{T|T}(x_T) \prod_{t=0}^{T-1} \mathsf{B}_{t,\theta}(x_{t+1}, x_t), \tag{3.17}$$

where $\mathsf{B}_{t,\theta}(x_{t+1}, \cdot)$ is the backward kernel, i.e. the distribution of X_t given X_{t+1} and $Y_{0:T}$. The backward kernel can be expressed as :

$$\mathsf{B}_{t,\theta}(x_{t+1}, x_t) = \frac{\phi^\theta_{t|t}(x_t) q_{\theta,t+1}(x_t, x_{t+1})}{\int \phi^\theta_{t|t}(u) q_{\theta,t+1}(u, x_{t+1}) du} \propto \phi^\theta_{t|t}(x_t) q_{\theta,t+1}(x_t, x_{t+1}).$$

Such a formula shows that $\phi^\theta_{0:T|T}$ may be decomposed using all the marginal filtering distribution $\phi^\theta_{t|t}$ from time step $t = 0$ to T. From a particle filter already performed on the entire dataset (i.e $\phi^\theta_{t|t}$ has been estimated from $t = 0$ until $t = T$ with a weighted sample $\{(\omega_t^i, \xi_t^i)\}_{i=1}^N$), we can easily construct a particle approximation of the backward kernel as follows :

$$\mathsf{B}^N_{t,\theta}(x_{t+1}, dx_t) = \sum_{i=1}^N \frac{\omega_t^i q_{\theta,t+1}(\xi_t^i, x_{t+1})}{\sum_{j=1}^N \omega_t^j q_{\theta,t+1}(\xi_t^j, x_{t+1})} \delta_{\xi_t^i}(dx_t). \tag{3.18}$$

Such a distribution can be used to generate states in the reverse time, from $t = T-1$ until $t = 0$, given future states, again with a SIR scheme. In particular, the **Forward Filtering Backward Smoothing(FFBS)** [69] algorithm performs this backward pass by keeping all particles fixed, but forgetting the genealogy given by the particle filtering weights, and computing new importance weights recursively. For any measurable function h on $\mathsf{X} \times \mathsf{X}$, the FFBS approximation of $\phi_{t-1:t|T}[h]$ is written :

$$\phi^{N,\mathsf{FFBS}}_{t-1:t|T}[h] = \sum_{i,j=1}^N \omega^{i,j}_{t-1:t|T} h(\xi^i_{t-1}, \xi^j_t), \quad \sum_{i,j=1}^N \omega^{i,j}_{t-1:t|T} = 1.$$

The new normalized weights $\omega^{i,j}_{t-1:t|T}$ for $1 \leq i,j \leq N$ and $1 \leq t \leq T$ are obtained recursively using (3.17) where the backward kernels are replaced by their particle approximations given by (3.18). At time T, define the smoothing weights by $\omega^i_{T|T} = \omega^i_T / \Omega^N_T$ for all $i \in \{1,\dots,N\}$, with Ω^N_T the sum of all filtering weights. For $t = T$ to $t = 1$,

$$\phi^{N,\mathsf{FFBS}}_{t-1:t|T}[h] = \sum_{i,j=1}^{N} \omega^j_{t|T} \frac{\omega^i_{t-1} q_{\theta,t}(\xi^j_{t-1}, \xi^i_t)}{\sum_{\ell=1}^N \omega^\ell_{t-1} q_{\theta,t}(\xi^i_{t-1}, \xi^\ell_t)} h(\xi^i_{t-1}, \xi^j_t),$$

which yields, for all $1 \leq i,j \leq N$,

$$\omega^{i,j}_{t-1:t|T} = \omega^j_{t|T} \frac{\omega^i_{t-1} q_{\theta,t}(\xi^j_{t-1}, \xi^i_t)}{\sum_{\ell=1}^N \omega^\ell_{t-1} q_{\theta,t}(\xi^i_{t-1}, \xi^\ell_k)} \quad \text{and} \quad \omega^i_{t-1|T} = \sum_{j=1}^N \omega^{i,j}_{t-1:t|T}.$$

In that scheme, each particle ξ^i_t at a given time t is reconnected to all possible ancestors at the previous time, as illustrated on the left figure of Figure 3.7.

The **Forward Backward Backward Simulation (FFBSi)** [104] does not compute new weights, but samples backward trajectories from $t = T$ to 0 among all the possible N^{T+1} trajectories that can be built from the particles $\{\xi^i_t\}_{i=1}^N$. Using the same decomposition as for the FFBS algorithm, each trajectory is sampled as follows. At time $t = T$, sample $J_T \in \{1\dots,N\}$ with probabilities $\omega^\ell_T / \Omega^N_T$, for $\ell \in \{1,\dots,N\}$. For $t = T-1$ to $t = 0$, J_t is sampled in $\{1\dots,N\}$ with probabilities :

$$\Lambda^N_t(J_{t+1}, i) = \frac{\omega^i_t q_{\theta,t+1}(\xi^i_t, \xi^{J_{t+1}}_{t+1})}{\sum_{\ell=1}^N \omega^\ell_t q_{\theta,t+1}(\xi^\ell_t, \xi^{J_{t+1}}_{t+1})}, \quad 1 \leq i \leq N. \tag{3.19}$$

These sampling steps are repeated independently N times to produce $\{J^\ell_0,\dots,J^\ell_T\}$, $1 \leq \ell \leq N$. Then, the smoothed expectation of any measurable function h on X^{T+1} is approximated by :

$$\phi^{N,\mathsf{FFBSi}}_{0:T|T}[h] = N^{-1} \sum_{\ell=1}^N h(\xi^{J^\ell_0}_0, \dots, \xi^{J^\ell_T}_T).$$

The backward process of FFBSi is illustrated on the right figure of Figure 3.7.

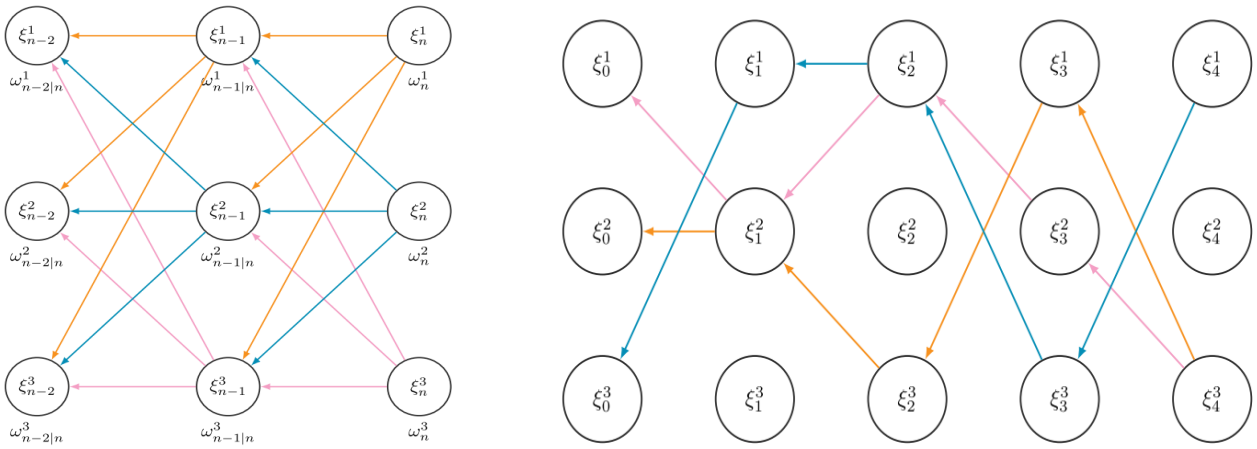


FIGURE 3.7 – Illustration of the forward recursion in FFBS (left), and FFBSi (right), for $N = 3$. **(Left) :** FFBS computes new backward resampling weights to reconnect particles to ancestors. Each color line represents the connection of a given particle to its three ancestors through the weights. **(Right) :** FFBSi forgets the filtering trajectories genealogy and samples new trajectories from the set of particles. Each color line represents a sampled backward trajectory.

Limits of existing smoothing algorithms. The smoothers based on forward-backward recursions, although leading to a much better approximation of the smoothing distribution than the PMS and coming with strong theoretical guarantees [59, 67, 71, 100], present several shortcomings. First, they are computationally costly. FFBS has a computational cost that grows to N^{T+1} when estimating $\phi^{\theta}_{0:T|T}$. The computational cost of FFBSi is lower, equal to N^2, yet remains prohibitive when using such algorithms on complex state-space models such as Deep Generative Models. It can be reduced to a complexity of order N but under conditions on the latent Markov chain and using an accept-reject procedure, see [67]. Secondly, they require the time horizon and all the observations to be available to perform the smoothing. Finally, they require the evaluation of the transition densities $q_{\theta,t}$, $1 \leq t \leq T$, and the observation densities $g_{\theta,t}$, $0 \leq t \leq T$; yet, for many state-space models, in particular generative models based on deep learning architectures, we do not have analytical expressions of such densities.

In chapter 7, we propose BIS, a new online smoothing algorithm (i.e the smoothing backward step does not requires the time horizon and all the observations), that uses a backward importance sampling step to significantly reduce the smoothing computational cost while providing efficient smoothing estimates.

1. When applied to a one-dimensional sine model, BIS performs similarly to the state-of-the-art online smoothing with an accept-reject mechanism [102], while having a computational complexity reduced by a factor of 10.
2. BIS can hence scale to high-dimensional models, and we demonstrate its ability to perform state estimation on a recurrent neural network.
3. BIS also makes use of pseudo-marginalisation techniques and may be used with random estimates of $q_{\theta,t}$, $1 \leq t \leq T$, and of $g_{\theta,t}$, $0 \leq t \leq T$, and hence is applicable on a broader scope of state-space models. The performance of our algorithm is assessed with a stochastic Lokta-Volterra model, where we combine Parametrix Estimators [6, 83] with the Wald Trick [80] to obtain an almost surely positive unbiased estimate of the transition density. This illustrates the applicability of BIS in very challenging settings.

Chapitre 4

Neural Language Models

4.1 Language Modelling and Natural Language Generation

Let $\mathcal{V} = \{w_i\}_{1 \leq i \leq |\mathcal{V}|}$ be a discrete vocabulary, i.e an ordered set of words of size $|\mathcal{V}|$. On $\mathcal{V}$, a statistical Language Model (LM) p_θ^{LM} assigns probabilities to a sequence of words $w_{1:T} = (w_1, w_2, ...w_T)$:

$$w_{1:T} \in \mathcal{V}^T \mapsto p_\theta^{LM}(w_{1:T}) \in [0,1]^T$$

Language Models are auto-regressive : this means that we can use the chain-rule of probability to factorize the joint probability of the sequence $p_\theta^{LM}(w_{1:T})$ into the product of probabilities over the next words w_t given the past words $w_{1:t-1}$:

$$p_\theta^{LM}(w_{1:T}) = p_\theta^{LM}(w_1) \prod_{t=2}^{T} p_\theta^{LM}(w_t | w_{1:t-1}) \tag{4.1}$$

Such a factorization allows to learn in practice a model $p_\theta^{LM}(w_t | w_{1:t-1})$ that predicts a next word given a sequence of past words. This reduces to a multi-class classification problem on sequential data, as words are sampled from a discrete vocabulary $\mathcal{V}$.

Conditional Language Models. The language model defined so far is called an unconditional language model, i.e it can generate text unconditionally, and does not require any context. Once the language model has been trained, at inference, an unconditional language model can generate a sequence of words starting from a special token, the "start of sentence" token, usually noted "SOS". This is referred to as open-ended text generation. The text generated will have a similar language distribution than the training dataset. For instance, it will resemble a news article if it has been trained on a news dataset, a encyclopedia article if it has been trained on Wikipedia data, a poem if has been trained on poetry data, etc. On the other hand, a *conditional* Language Model assigns probabilities to a sequence of words given a specific context c, and can be defined as $p_\theta^{LM}(w_{1:T}|c)$. The context can be any kind of data structure,

such as an image, a video, and input text sequence, etc.

In particular, when the context is a words sequence, the conditional language model is a sequence-to-sequence model, as it takes as input a sequence, and outputs another sequence. The most famous application of such models is Machine Translation [32, 147, 181], that takes as context a input sentence in a source language and outputs the translated sentence in a target language. Deep-Learning based approaches for such LM are based on encoder-decoder frameworks, as described in Section 3.2.1.

In the following sections, we will note $p_\theta^{LM,(c)}$ a generic language model, that could be either an unconditional language model, or a conditional language model.

Natural Language Generation designs any setting in which we generate (i.e write) new text, by using a language model $p_\theta^{LM,(c)}$. It corresponds to the inference phase of a Language Modelling task. Once we have learned a set of parameters $\widehat{\theta}$ on the training dataset, we are usually interested in using $p_{\widehat{\theta}}^{LM,(c)}$ for generating new text, given for instance the context c for conditional language models, or given an history of past words for unconditional language models.

Formally, let $(w_{0:t}, (c))$ be the input of our learned language model for generating text, i.e a sequence of past words (that could simply be the start token "SOS"), and an optional context. When performing natural language generation, we aim to generate a sequence $w_{t:t+H}$ of new words. Algorithms that performed such operation from a language model, are called decoding algorithms. They decode sequentially a text sequence, by selecting sequentially one word at each timestep i of the generation process. The most common decoding algorithms are detailed hereafter.

Greedy decoding consists in selecting the greedy token, i.e the token with the highest probability from the language model distribution. Formally, greedy decoding performs at each time step i of the text generation process the following operation : $w_{t+i} = \text{argmax}_{w \in \mathcal{V}} p_{\widehat{\theta}}^{LM,(c)}(w|w_{1:t-1+i}, (c))$

Sampling decoding samples at each timestep of the text generation process from the multinomial distribution over the vocabulary given by the language model : $w_{t+i} \sim p_{\widehat{\theta}}^{LM,(c)}(.|w_{1:t-1+i}, (c))$

These two basic decoding algorithms usually lead to text degeneration issues in the decoded text. Greedy decoding tends to generate repetitive and generic responses, while sampling decoding tends to generate incoherent text. More advanced text decoding techniques have thus been developed to give a better quality/diversity trade-off in the language being generated.

Sampling with temperature decoding adds a temperature parameter τ to modify the language model distribution, before sampling from it. Formally, let $f_{\widehat{\theta}}^{\text{LM},(c)}$ be the unnormalized log distribution of the language model. Then, at iteration i of the decoding process, w_{t+i} is selected as follows :

$$w_{t+i} \sim \text{softmax}\left(\frac{f_{\widehat{\theta}}^{\text{LM},(c)}(w_{1:t-1+i})}{\tau}\right).$$

The temperature τ is usually chosen to be less than one, so that the language model initial distribution is transformed into a peakier distribution thus artificially decreasing the relative probability of tail words and reducing the incoherence in text generation produced by basic sampling decoding.

Beam-Search decoding [236, 91] searches over a beam B of most probable words at each time step of the generating process, to get an estimate of the most likely sequence of words. At timestep $i-1$ of the decoding process, beam-search maintains a beam of K hypotheses $\mathcal{W}^K_{t-1+i} = \{(w^k_{0:t-1+i})\}_{1\leq k\leq K}$ that maximizes the log-likelihood score : $s(w^k_{0:t-1+i}) = \log p^{LM,(c)}_{\widehat{\theta}}(w^k_{0:t-1+i})$. The beam is expanded at the next decoding time step i by first selecting the top K next words $\{w^{k,k'}_{t+i}\}_{1\leq k'\leq K}$, for each element of the beam $(w^k_{0:t-1+i})$. This leads to the construction of K^2 new hypotheses, for which we compute a new score as follows :

$$s(w^k_{0:t-1+i}, w^{k,k'}_{t+i}) = s(w^k_{0:t-1+i}) + \log p^{LM,(c)}_{\widehat{\theta}}(w^{k,k'}_{t+i}|w^k_{0:t-1+i}).$$

Using the updated score, the algorithm then selects the top K sequences at timestep i, leading to a new beam $\mathcal{W}^K_{t+i}$ of size K.

Top-k sampling first selects the top-k words given by the language model distribution, and then samples on the rescaled probability distribution restricted on these top-k words.

Nucleus sampling [127] reshapes the language model distribution to be sampled from, by restricting it to the most likely tokens that takes up the vast majority of its probability mass. It considers the smallest set of words $V^{(\bar{p})}$ contained in a mass $\bar{p}$ of the initial LM distribution. Formally, $V^{(\bar{p})}$ is defined as the smallest subset V' of $\mathcal{V}$ such that $\sum_{w\in V'} p^{LM,(c)}_{\widehat{\theta}}(w|w_{1:t-1+i}) \geq \bar{p}$. A rescaled distribution computed from a softmax restricted to $V^{(\bar{p})}$ is then used to sample the next token w_{t+i}.

Greedy	*I enjoy going running in the morning*. I like to run in the afternoon. I like to run in the evening. I like to run in the morning. I like to run in the evening. I like to run in the evening. I like
Sampling	*I enjoy going running in the morning*. I've always done that, but I don't let you do it. Not if I'm around. Soowell also made the comments that Dale Francona's interview with the author is a cal
Beam-Search	*I enjoy going running in the morning*. I love running in the morning. I love running in the morning. I love running in the morning. I love running in the morning. I love running in the morning. I love running in the morning.
Top-k sampling	*I enjoy going running in the morning*. I think it's a good thing to have something out and to have the opportunity to be around people who are different from myself and be there for them when they need it. Bournier is."
Nucleus sampling	*I enjoy going running in the morning*. I don't know if I am even going to run for a day. But I have no idea how to run in these conditions. I want to live. The next day, the New York Post

TABLE 4.1 – Example of different decoded text outputs for the input sentence "I love running in the morning.", for the decoding procedures detailed in this section.
The Language Model used for generating text is OpenAI GPT-2 [232]. Beam-Search uses 5 beams, top-k sampling uses 50 top-k words, and nucleus sampling uses a nucleus mass $\bar{p}$ equal to 0.95.

Limits of decoding techniques for natural language generation. Today, Sampling with temperature, top-k sampling, and nucleus sampling are the most popular decoding approaches. They exhibit language performances (when looking at the trade-off quality/diversity of text samples) similar to beam-search, while being easier to implement, and computationally more efficient. Yet, Figure 4.1 shows that there is no perfect decoding algorithm. It displays different text continuation samples for the input sentence "I enjoy running in the morning." using the different decoding techniques. Greedy and beam-Search decoding, the two methods based on maximizing the log-likelihood score, generate syntactically coherent text, but made of repetitive text sequences. Sampling decoding generates incoherent and uncorrect text. In between, top-k sampling and nucleus sampling generate better alternative of text samples ; yet, they tend to lose meaning and coherence, once the text sequence gets longer ("I want to live. The next day, the New York Post...", and "Bournier is" does not bear any relation with the input text sequence). In today common text decoding methods, increasing diversity comes at the expense of language quality.

The Sequential Monte Carlo Transformer developed in Chapter 6 proposes another way to model diversity in sequential data generation, by relying on stochastic self-attention parameters.

1. It relies on a stochastic self-attention model within a Transformer network : the attention parameters described in Equation 3.6 become stochastic latent states that model the variability of the observations (the network input data). Attention parameters (keys, queries, values) are not observed anymore as a random noise is introduced in the self-attention transformations.
2. In doing so, the objective function to train the network (the log-likelihood of the observation) becomes intractable. Using Fisher's Identity, we propose to use a SMC particle filter algorithm (see Section 3.3.1) to compute an estimate of the score function and run a stochastic gradient descent algorithm.
3. In a regression setting, the model outputs a predictive distribution, instead of a single-point estimate of a given target, thus allowing to quantify uncertainty in sequential data problems, such as multi-step time-series forecasting.
4. In a Language Generation setting, representing variability in language data by a rich latent structure could help solving the quality/diversity trade-off when generating text at inference.

4.2 SL-based Language Models

The factorization of Equation 4.1 allows to learn in practice a model $p_\theta^{LM,(c)}(w_t|w_{0:t-1})$ that predicts a next word given a sequence of past words. This can be seen as a multi-class classification problem on sequential data, as words are sampled from a discrete vocabulary $\mathcal{V}$. Hence, learning statistical Language Models can be modelled as

a sequential Supervised Learning (SL) Problem, where the target is the next word w_t, and the inputs are the past sequences of words $w_{0:t-1}$. In practice, when learning Language Models with SL on one or several text datasets, we follow the following steps.

— Extracting a processed dataset $\mathcal{D} = \{(w_1^i, ..., w_T^i)\}_{i=1}^n$ made of n samples of text sequences $(w_1^i, ..., w_T^i)$ of same length T, with w_t^i being a word. The lengths of the original text sequences may vary, as the number of words in a sentence, paragraph, question, caption vary. Thus, the first processing step is to transform the original dataset into samples of words sequence of fixed lengths. This is done by padding or truncating the words sequences. Given a maximal sequence lengths T, the short words sequences are completed with a special padding token (not accounted in the loss function) from the vocabulary, while the longer sequences are truncated.

— Build a vocabulary $\mathcal{V}$ that defines the set of words available for the language model, and corresponds to the number of classes in the sequential classification task. The vocabulary maps words to ids, and is constructed either using all the different words in the text dataset, or selecting words whose number of occurrences in the text dataset exceeds a predefined threshold. In that case, non-selected words are assigned the special unknown token "UNK". Modern vocabulary designs [252, 291] for large-scale text datasets include sub-words units : the combination of such units can cover multiple words, in order to reduce the vocabulary size.

Given a processed dataset $\mathcal{D}$ and its associated vocabulary $\mathcal{V}$, the objective function $J(\theta)$ to train $p_\theta^{LM,(c)}$ is equivalent to the categorical cross-entropy between the LM distribution and the ground-truth next token w_t :

$$J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{t=1}^{T}\sum_{m\in\mathcal{V}} \mathbb{1}_{w_t^i=m} \log p_\theta^{LM,(c)}(w_t^i = m|w_{1:t-1}^i). \tag{4.2}$$

Learning SL-based Language Models with Transfer Learning. As explained in the introduction, most of today state-of-art conditional language models leverage the transfer learning abilities of pretrained Language Models (PLM), such as the GPT [230], GPT-2 [232] and GPT-3 [33] variants of OpenAI. Instead of learning from scratch the downstream task of interest (for example, Machine Translation), they are generally fine-tuned from the pretrained language model, i.e. they are initialized with the PLM weights, and further trained for a few epochs on the downstream task. Other transfer learning methods to adapt a PLM to a downstream task include few-shot learning [232, 33] (the PLM is only given few demonstration samples as an input prompt to steer the language generation towards the task objective), and prompt tuning [182, 227, 115, 172, 163] (instead a fine-tuning the PLM weights, a continuous prompt as the task-specific input for the PLM is learned on the task dataset).

Limits of SL-based Language Models. First, SL-based Language Models require supervised datasets. While these datasets come almost for free for unconditional LM (mostly constituing of web scrapes), for conditional language models, they are costly to collect. Indeed, Machine Translation, Text Summarization or Image Captioning systems need hand-labelled data. Secondly, as mentioned in the introduction, they suffer from the exposure bias [18], and also tend to inherit the language bias present in the supervised dataset [243]. Recently, although transfer learning approaches from generic pretrained LM have achieve state-of-the-art results on many NLU and NLG tasks [65, 230, 165], they suffer from their own pitfalls [28]. First, fine-tuning big pretrained LM suffer from instability [65, 160, 199, 233] (training the same model with multiple random seeds can lead to a large variance of the downstream task performance), and sometimes catastrophic forgetting [45] : the pretrained model forgets previously learned knowledge and overfits to target domains. Additionally, some researchers have pointed several important weaknesses of pretrained Language Models, mainly their vulnerability to adversarial attacks [135, 279], their intrinsic biases leading to harmful behaviors on minority groups [26, 148], and their lack of interpretability [28] : there is no real understanding about what a pretrained model is capable of doing, why it outputs certain behaviors, and finally how it does it.

In Chapter 5, we propose TrufLL, a Reinforcement Learning algorithm to learn a conditional Language Model. Such a LM, whose probability distribution is not learned by explicitly targeting ground-truth text samples, might avoid the repetitiveness found when generating text from LMs trained with Supervised Learning.

1. TrufLL proposes a truncation algorithm to truncate the large vocabulary action space of the RL language agent using a pretrained language model. By doing so, on a NLP point of view, it provides a generic linguistic prior knowledge to the Language agent. On a RL point of view, it significantly reduces its effective action space, hence allowing to learn a policy Language Model "from scratch", i.e without requiring human-labelled datasets.
2. By doing so, it avoids the typical language biases found in SL-based Language Models, and outputs an original language distribution compared to the initial human-labelled dataset.

4.3 RL-based Language Models

Text generation can be naturally framed as a sequential decision making problem, with the sequence of words seen as successive actions over a vocabulary. Thus, learning a language model can be cast as a Reinforcement Learning (RL) [266] problem. In Reinforcement Learning, an agent interacts with an environment, by taking successive actions and moving into states, and receiving a scalar feedback, called the reward. Reinforcement Learning consists in learning the optimal behavior for the agent with respect to the environment, i.e the optimal mapping between states and actions which maximizes the expected sum of rewards. Figure 4.1 illustrates the Reinforcement

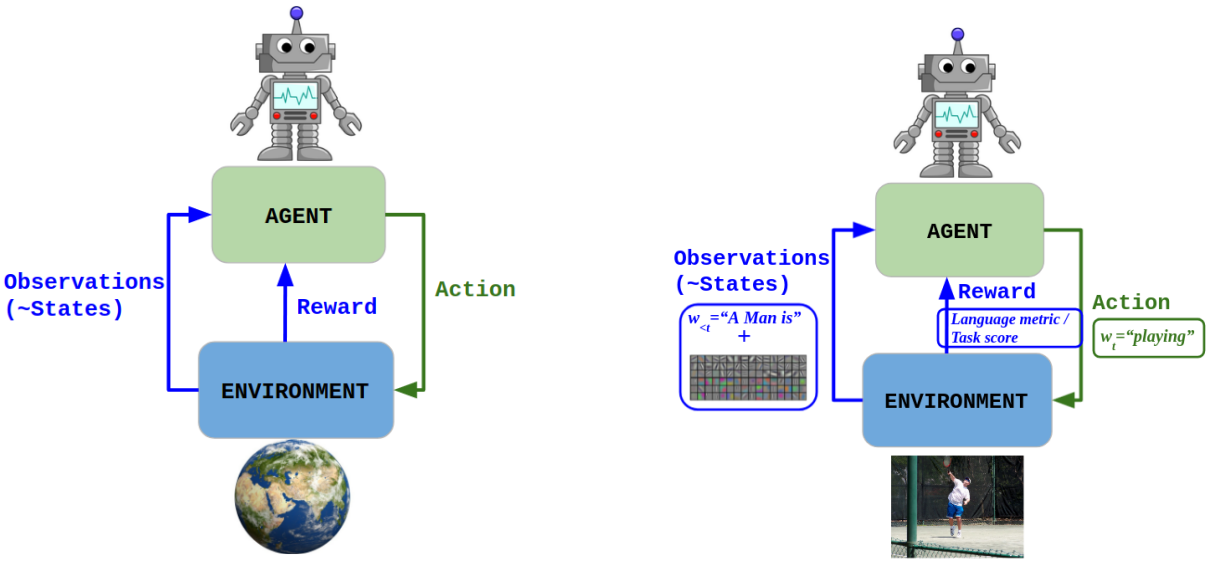


FIGURE 4.1 – (Left) : Illustration of the RL Framework. (Right) : Example of a Language Generation task cast as a RL problem : here, the Language Model agent tries to generate a caption on a given image.

Learning framework (left figure) and applies it to Language Generation (right figure), by taking the example of an image captioning task. In a Language Generation setting, the Agent is the conditional Language Model to be learned. Its states at a given time step t is the past sequence of words $w_{<t} = (w_1, .., w_{t-1})$ it already uttered and the context c (in the image captioning example, the images features). Its next actions is the next word w_t to be uttered. The reward received could be a language metric (such metrics are detailed in Section 4.4) evaluating the sequence of words being uttered, or simply a task score, that in the example would assess whether the caption corresponds to the image.

The next paragraphs describe the mathematical framework and the Reinforcement Learning algorithms to learn such RL-based Language Models.

Markov Decision Processes. A Markov Decision Process (MDP) formally describes a reinforcement learning environment. A MDP $\mathcal{M}$ is a 5-tuple (S, A, P, r, γ), whose five elements are described below.

— S is the set of states, it is either a finite discrete set, or is made of continuous elements (in that case S is a bounded subset of $\mathbb{R}^d$). In a Language Generation setting, the state at a given time step t is written $s_t = (w_{<t}, c)$ and is made of the embeddings of the past sequence of words $w_{<t}$, and the context c (an image, an input text sequence, ...) that grounds and conditions the LM.

— A is the set of possible actions a that can be performed in every state s of S. Similarly, A is either a finite discrete set, or is made of continuous elements, in that case A is a bounded subset of $\mathbb{R}^d$. In a Language Generation Setting, the Action Space is the Vocabulary $\mathcal{V}$ of possible words to be uttered by the Language Agent.

— $r : S\text{x}A \mapsto \mathbb{R}$ is the reward function providing the reward $r(s, a)$ when being in state s, and taking action a. At time step t, we will denote $r_t = r(s_t, a_t)$ the reward received by the RL agent. Rewards for RL-based Language Models are usually language metrics (evaluating the quality of the language being generated), or a task score, see Chapter 5.

— $P : S\text{x}A\text{x}S \mapsto [0, 1]$ is the state transition distribution, where for all time step t, all $s, s' \in S$ and all $a \in A$, $P(s_{t+1} = s'|s_t = s, a_t = a)$ is the probability of reaching s', when being in state s and taking action a.

— $\gamma \in]0, 1]$ is the discount rate, which weights the importance of future rewards. The parameter γ penalizes rewards received far away in the future, to model the future uncertainty, and the human preferences for immediate rewards. Yet, in a Language Generation setting, we usually choose $\gamma = 1$.

A Markov Decision Process is considered to formally describe a reinforcement learning problem, when assuming the following [267].

— The environment is fully observable, i.e the observations of the environment correspond to the agent states.

— The transition dynamics follows the Markov Property, i.e the probability of reaching a state s_{t+1} at time step $t + 1$ given past states only depends on the previous state s_t only, and not all the previous states.

Besides the five elements describing a MDP, to fully understand a RL problem, it is useful to introduce the following quantities.

— **The agent policy** denoted by π maps a given state to an action. It describes formally the behavior of the agent, when interacting with its environment. A policy can be either deterministic, i.e $a = \pi(s)$ is a deterministic quantity, or stochastic, i.e a random variable (in that case, we note $a \sim \pi(\cdot|s)$). In a Language Generation setting, the agent policy is exactly the conditional Language Model we would like to learn. When referring to a parametric policy with parameters θ, we will note it π_θ.

— **The return** The return is given by $G_t = \sum_{k\geq 0} \gamma^k r(s_{t+k+1}, a_{t+k+1})$ and is the total discounted cumulative reward received by the agent from timestep t when following a policy π.

— **The Value Functions** V_π and Q_π to evaluate how good is a policy with respect to the expected return it provides. The State Value Function V_π measures the benefit of being in a state s, and the Action-State Value Function Q_π, measures the benefit of being in a state s and taking action a. They are defined as follows :

$$V_\pi : S \mapsto \mathbb{R}; \quad V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) \middle| s_t = s \right], \tag{4.3}$$

$$Q_\pi : S\text{x}A \mapsto \mathbb{R}; \quad Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k r(s_{t+k+1}, a_{t+k+1}) \middle| s_t = s, a_t = a \right]. \tag{4.4}$$

Policy Gradient Methods. There are two main types of algorithms to solve a MDP, i.e. to find the optimal policy π^* which maximizes the expected return of the agent. *Value-Based RL* makes use of the Value Functions and the Bellman Equations [267] to learn the optimal value function, from which the optimal policy is derived. *Policy-based*

RL directly learns a parametric policy π_θ. Although it suffers generally from sample inefficiency and high-variance, Policy-Based RL (also referred to as Policy Gradient methods) scales to high-dimensional and continuous action spaces, and allows to learn a stochastic policy. It is hence the privileged set of methods to learn RL-based Language Models.

Policy Gradient methods find the best set of parameters θ^* given a stochastic parametric policy $\pi_\theta(a|s)$. To do so, we need to define an objective function $J(\theta)$ which evaluates the quality of the policy π_θ. As solving a reinforcement learning task consists in finding the optimal policy which maximizes the agent expected return, objectives functions are obviously based on this expected return. In episodic environments, we generally use the expected return from the initial state :

$$J(\theta) = \mathbb{E}_{a_t \sim \pi_\theta(.|s_t)} \left[\sum_{t=0}^{T} \gamma^t r(a_t, s_t) \middle| s_0 \right], \tag{4.5}$$

where T corresponds to the time step for which a terminal state s_T is reached. In a Language Generation setting, we will consider episodic environments : the terminal state happens when a special token is generated, or when a maximum text length is reached. In such a setting, the sum in Equations 4.3 and 4.4 becomes a finite sum from $t = 0$ to $t = T$. Similarly, the return from timestep t is a sum of cumulative rewards until timestep T.

The **Policy Gradient Theorem** expresses $\nabla_\theta J(\theta)$ as a function of the policy π_θ. For any differentiable policy $\pi_\theta(a|s)$, the policy gradient is :

$$\nabla_\theta J(\theta) = \mathbb{E}_{a_t \sim \pi_\theta(.|s_t)} \left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) Q_{\pi_\theta}(s_t, a_t) \right]. \tag{4.6}$$

The expectation is approximated using a Monte Carlo approach, i.e. using an empirical average over a finite batch of trajectories. A trajectory is a succession of transitions $\big(s_t, a_t \sim \pi_\theta(.|s_t), r_t = r(a_t, s_t)\big)$ from an initial state s_0 to a terminal state s_T. Policy Gradient algorithms use the Policy Gradient Theorem and compute an estimation of $\nabla_\theta J(\theta)$. Several variants exist that estimate differently $Q_{\pi_\theta}(s_t, a_t)$.

The vanilla REINFORCE algorithm [286] uses the return $G_t = \sum_{k \geq 0} \gamma^k r_{t+1+k}$ averaged over a certain number of trajectories $\tau \in \Upsilon$, i.e. sequences of states and actions sampled conditionally on (s_t, a_t), as an unbiased estimation of $Q_{\pi_\theta}(s_t, a_t)$. This yields the following estimation of $\theta \mapsto \nabla_\theta J(\theta)$:

$$\theta \mapsto \frac{1}{|\Upsilon|} \sum_{\tau \in \Upsilon} \sum_{t=0}^{T} [\nabla_\theta \log \pi_\theta(a_t|s_t) G_t^\tau], \tag{4.7}$$

where G_t^τ is the return of the trajectory τ, and $|\Upsilon|$ is the number of sampled trajectories. This algorithm suffers from high-variance. A simple extension to REINFORCE reducing variance subtracts a baseline b to the return, where b is a function that depends only on the current state. At timestep t, the advantage function is defined as $A_t = G_t - b(s_t)$

and the estimation of the gradient of J becomes

$$\theta \mapsto \frac{1}{|\Upsilon|} \sum_{\tau \in \Upsilon} \sum_{t=0}^{T} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) A_t^\tau \right], \tag{4.8}$$

where A_t^τ is the advantage function for trajectory τ. A parametric estimator $V_\phi(s_t)$ of the Value Function is usually taken as the baseline b and is learned by minimizing the mean squared error between $V_\phi(s_t)$ and G_t.

Finally, **Actor-Critic algorithms** [283, 114] estimate $Q_{\pi_\theta}(s_t, a_t)$ with a parametric estimator with parameters W learned simultaneously with policy π_θ.

Most existing RL-based Language Models simply use a REINFORCE with baseline algorithm to optimize the Language Model policy. In Chapter 5, we propose to use instead Trust-Region Methods [247], more precisely the Proximal Policy Optimization (PPO) algorithm [249]. Trust Region Methods optimize a surrogate objective, subject to a constraint on the size of the policy update. Such constraint allows smoother policy updates, resulting in an optimization algorithm having a lower variance and better convergence rate than REINFORCE. The surrogate objective is the following :

$$\underset{\theta}{\text{maximize}} \quad \mathbb{E}_{a_t \sim \pi_{\theta_{old}}(.|s_t)} \left[\sum_{t=0}^{T} \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t \right], \tag{4.9}$$

$$\text{subject to} \quad \mathbb{E} \left[\sum_{t=0}^{T} \mathrm{KL}\big(\pi_{\theta_{old}}(.|s_t), \pi_\theta(.|s_t)\big) \right] \leq \delta\,, \tag{4.10}$$

where θ_{old} are the parameters before the update, A_t is the advantage function as previously defined, and KL is the Kullback-Leibler divergence between two distributions. Let $\rho_t = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$ be the ratio between the old policy and the one being updated. PPO, instead of using a hard constraint, optimizes the policy with the following clipped surrogate objective :

$$L_{PPO}(\theta) = \frac{1}{|\Upsilon|} \sum_{\tau \in \Upsilon} \left[\sum_{t=0}^{T} \min\left(\rho_t A_t^\tau, \mathrm{clip}\big(\rho_t, 1-\epsilon, 1+\epsilon\big) A_t^\tau \right) \right], \tag{4.11}$$

where A_t^τ is the advantage function for trajectory τ, $|\Upsilon|$ is the number of trajectories, ϵ is a small value, usually taken between 10^{-1} and 10^{-2}, and

$$\mathrm{clip}(x, a, b) = \begin{cases} x & \text{if } x \in [a, b]\,, \\ a & \text{if } x < a\,, \\ b & \text{if } x > b\,, \end{cases}$$

is the function that clips x in interval $[a, b]$.

Exploration in Natural Language Action Spaces. In Reinforcement Learning, a key challenge is the Exploration/Exploitation dilemma. To obtain a lot of reward, a reinforcement learning agent must prefer actions that it has tried in the past and found to be effective in producing a good reward. But to discover such actions, it has to try actions that it has not selected before. The agent has to *exploit* what it already knows in order to obtain reward, but it also has to *explore* in order to make better action selections in the future.

In small discrete action spaces or continuous action spaces, exploration techniques such as ϵ-greedy policies, Boltzmann exploration [2], entropy regularization [196], or noise perturbation [92] have been designed to solve the Exploration/Exploitation dilemma. Yet, in RL-based Language Models, the action space is the vocabulary, i.e a large discrete action space made of more than ten thousands words. Designing efficient exploration techniques for Natural Language action spaces remains a key research problem. Most existing RL-based LM avoids the problem, by including a pre-training phase for the Language Model policy. In order to be able to form intelligible language that could result in some positive reward signal, the policy is pretrained on the task dataset with a supervised learning objective, and is then fine-tuned on the task at hand with RL.

In Chapter 5, we propose a RL algorithm for Natural Language action spaces that leverages the structure of language to drive the exploration process of the Policy Language Model. The algorithm enables to learn a RL-based Language Model *from scratch*, i.e without the pre-training phase on the supervised dataset.

Limits of existing approaches for RL-based Language Models. The pre-training with SL and fine-tuning with RL approach that characterizes today RL-based Language Models have several pitfalls. First, the pre-training phase requires human-labelled datasets, that are costly to collect. Secondly, the change in the training objective between the pre-training and fine-tuning phases induce the language drift phenomenon [262, 185, 155], i.e the language generated by the agent drifts semantically and syntactically from natural language.

In Chapter 5, the TrufLL algorithm, for TRUncated ReinForcement Learning for Language, allows to get rid of the pretraining phase in RL-based Language Models.

1. TrufLL truncates the action space of the RL Language Agent to a small number of grammatically plausible words by using a pretrained language model (GPT-2 [232]).
2. TrufLL constitutes an original and novel way to combine RL-based Language Models with pretrained Language Models for conditional language modelling.
3. When applying TrufLL on Visual Question Generation tasks, the promising results show TrufLL is a first step for Learning RL-based Language Models from scratch, even on large vocabularies (more than 10k words), and complex language generation tasks.

4.4 Evaluation Metrics for Language Models

Evaluating a language model is a challenging problem. Designing automatic metrics that assess the language quality of a LM as humans would do remains a key research problem in NLP. When evaluating Natural Language Generation, researchers usually use several metrics, as well as a qualitative assessment of the text samples generated, to get a comprehensive overview of how performant is a language model. The choice of metrics depends usually on the natural generation task being considered. Some metrics have indeed been designed with in mind the evaluation of a specific NLG task (for instance Machine Translation, or Text Summarization), and are more relevant for specific language distributions (short sentences, paragraphs, etc.). We detail the language metrics used in the research works of this thesis, and refer to section 7 of [97] and Section 4 of [218] for a more comprehensive overview of evaluation methods for Language Modelling.

Evaluating Language Quality. The most popular metric to evaluate a language model $p_\theta^{LM,(c)}$ is its **perplexity** on the test set. The perplexity is defined as the inverse probability of the language model on the test set, normalized by the number of words. For a test dataset $\mathcal{D}_{\text{test}} = (w^1, w^2, ...w^{\bar{N}})$ with $\bar{N}$ words, the perplexity $\text{ppl}(\mathcal{D}_{\text{test}})$ is computed as follows :

$$\text{ppl}(\mathcal{D}_{\text{test}}) = \sqrt[\bar{N}]{\frac{1}{p_\theta^{LM,(c)}(w^1, w^2, \dots, w^{\bar{N}})}} = \sqrt[\bar{N}]{\frac{1}{\prod_{i=1}^{\bar{N}} p_\theta^{LM,(c)}(w_i|w_{1:i-1})}}. \tag{4.12}$$

Note that the perplexity is the exponential of the cross-entropy between the language model and the test data distribution. The lower it is, the better is considered to be the language model. Indeed, a low perplexity implies generating with a high-probability the text samples of the test set. As the perplexity is related to the language model distribution over a vocabulary $\mathcal{V}$, perplexities of different language models are comparable only over the same vocabulary, and the same test dataset.

The perplexity is an *intrinsic* evaluation metric, i.e it measures the quality of a language model, independently of any application. It is thus particularly relevant for unconditional language models, that perform open-ended natural language generation.

For conditional language models, evaluation methods include n-gram based metrics comparing the similarity of the text samples generated by the LM with ground-truth text samples. We will refer to the former as *candidates sentences* and note the set $\mathcal{C}$, and the latter as *reference sentences* and noted the set $\mathcal{R}$. A n-gram is a text segment (i.e consecutive words in a text sequence) made of n words. We detail hereafter three common n-gram based metrics, the BLEU score, the METEOR score, and the CIDEr score.

The BLEU score [214], for BiLingual Evaluation Understudy, has been originally designed to measure the accuracy of Machine Translation systems. A text sample from a language model is matched with several reference text samples. Scores are computed and averaged by considering the presence of several n-grams from the candidate

sentences in the reference sentences. The BLEU score, given a set $\mathcal{C}$ of candidates sentences and a set $\mathcal{R}$ of references is computed as follows :

$$\text{BLEU}(\mathcal{C},\mathcal{R}) = \text{BP}.\exp\big(\sum_{n=1}^{N}\omega_n \log p_n(\mathcal{C},\mathcal{R})\big); \quad p_n(\mathcal{C},\mathcal{R}) = \frac{\sum_{s\in\mathcal{C}}\sum_{\text{ngram}\in s}\mathbb{1}_{\text{ngram}\in\mathcal{R}}}{\sum_{s\in\mathcal{C}}N^s_{\text{ngram}}} \tag{4.13}$$

where p_n is the modified precision score counting matching n-grams for all the candidate sentences, $\mathbb{1}$ is the indicator function, and N^s_{ngram} is the number of n-grams in sentence s. w_n is a weight weighting the importance of each n-gram n : the most common BLEU score computes matching n-grams for $n \in \{1,...,4\}$, and all weights w_n are equal to 0.25. BP is a brevity penalty factor computed over the entire test dataset, to penalize shorter candidates sentences in $\mathcal{C}$. The major pitfall of the BLEU score is that it only matches n-grams and does not consider the overall syntactic correctness of the text sample.

The METEOR score [16], for Metric for Evaluation of Translation with Explicit ORdering, is based on an explicit word-to-word matching between a candidate sentence and reference sentences. The matching function uses stemming techniques, to match not only exact words but also words that are morphological variants or synonyms of each other. This metric has been designed to address the weaknesses of the BLEU score, mainly the fact that the latter does not take in account the recall for n-gram matching, i.e the proportion of the matched n-grams out of the total number of n-grams for the reference sentences. To compute the METEOR score, we first compute the precision $p_1(\mathcal{C},\mathcal{R})$ and recall $R_1(\mathcal{C},\mathcal{R})$ for matched unigrams.

$$\text{METEOR} = \frac{10p_1(\mathcal{C},\mathcal{R})R_1(\mathcal{C},\mathcal{R})}{R_1(\mathcal{C},\mathcal{R}) + 9p_1(\mathcal{C},\mathcal{R}}) * (1 - \text{Penalty})$$

where Penalty is a penalty term to penalize shorter sentences.

The CIDEr score [278], for Consensus-based Image Description Evaluation, has been initially designed for evaluating image descriptions. The computation of the score is done after applying stemming and representing every text sequence as a set of unigrams to four-grams. The cosine similarity is used to match the n-grams from the candidate sentences to the ones from the references, and the final score gives less weight to most common n-grams. For n-grams of length n, the CIDEr_n score is computed using the average cosine similarity between the TF-IDF weighting [186] vectors of the candidate sentence c_i and the TF-IDF weighting vectors of the set of reference sentences $R_i = \{r_{ij}, j = 1,...m\}$:

$$\text{CIDEr}_n(c_i,R_i) = \frac{1}{m}\sum_{j=1}^{m}\frac{g^n(c_i) * g^n(r_{ij})}{||g^n(c_i)||g^n(r_{ij})||}; \qquad \text{CIDEr}(c_i,R_i) = \sum_{n=1}^{4}\omega_n\text{CIDEr}_n(c_i,R_i)$$

where $g^n(c_i)$ (resp. $g^n(r_{ij})$) are the vectors formed by the TF-IDF weighting $g_k(c_i)$ (resp. $g_k(r_{ij})$) for all n-gram of length n, w_n are weights for each n-gram, and $||.||$ designs the norm of the vector.

Evaluating Language Diversity. In open-ended NLP domains such as conversational models or creative writing systems, successful language models should be able to produce a diverse range of high-quality text samples, rather than merely one single output text sequence. For instance, when considering a conversational language model, we would rather avoid generic and uninformative responses, such as "I don't know". In such setting, the model should be able to conserve about a wide variety of topics even if it sometimes imply odd remarks, rather than frequently repeating the safest response over and over. This is why evaluating diversity in Natural Language Generation is important; in open-domain NLG systems, such evaluation is complementary to language quality evaluation. Successful Language Models have a good quality/diversity trade-off [301].

The **self-BLEU score** was introduced in [307], as a measure of diversity in language generation, and has been originally applied to detect mode collapse in Generative Adversarial Networks (GAN) Language Models [54]. This metrics compute a BLEU score between several text samples from a Language Model, to mesure how similar are the samples. Given a set of N text samples, the metric computes first N BLEU scores, by taking each sample as the candidate sentence, and the remaining ones as reference sentences. The final self-BLEU score is the average of the N BLEU scores. The lower the self-BLEU score, the more diverse are the text samples.

Chapitre 5

Learning Natural Language Generation from Scratch with Truncated Reinforcement Learning

This chapter introduces TRUncated ReinForcement Learning for Language (TrufLL), an original approach to train conditional language models from scratch by only using reinforcement learning (RL). As RL methods unsuccessfully scale to large action spaces, we dynamically truncate the vocabulary space using a generic language model. TrufLL thus enables to train a language agent by solely interacting with its environment without any task-specific prior knowledge ; it is only guided with a task-agnostic language model. Interestingly, this approach avoids the dependency to labelled datasets and inherently reduces pretrained policy flaws such as language or exposure biases. We evaluate TrufLL on two visual question generation tasks, for which we report positive results over performance and language metrics, which we then corroborate with a human evaluation. To our knowledge, it is the first approach that successfully learns a language generation policy (almost) from scratch. [1]

5.1 Introduction

Since the development of generic language models trained on massive unlabelled text corpora [232, 33], state-of-the art language processing systems rely on *sequential transfer learning* [242]. The pretrained Language Model (LM) is fine-tuned on the downstream task using a standard supervised learning (SL) objective [290, 222]. Yet, such an approach suffers from several issues [45] : (i) catastrophic forgetting when a model forgets previously learned knowledge and overfits to target domains, (ii) computational inefficiency from fine-tuning billion-parameters

1. Code is available at https://github.com/AMDonati/RL-NLP

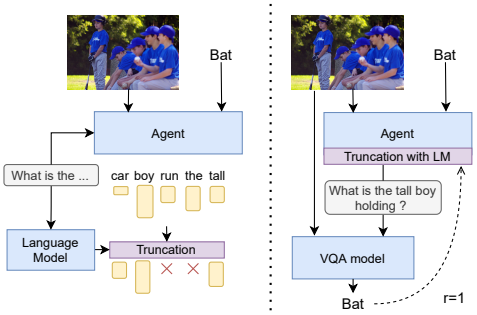


FIGURE 5.1 – (left) In a conditional language generation task as VQG, TrufLL truncates the vocabulary space by using a language model. Here, 'run,' and 'the' are syntactically incorrect and thus truncated. Yet, 'car' is not trimmed as the LM is not visually grounded. (right) In a VQG training loop, the agent generates a question given an image-answer pair, which is then fed to a VQA model predicting an expected answer. If both answers match, the agent is rewarded.

networks, and (iii) the need of supervised datasets. Moreover, task-specific language models learned with SL suffer from well-studied text degeneration issues [127], such as the exposure bias [18], language biases [243, 134], or a lack of diversity [168].

On the other hand, text generation can be naturally framed as a sequential decision making problem, with the sequence of words seen as successive actions over a vocabulary. Thus, some researchers have recently focused on learning language models using instead Reinforcement Learning (RL) [262, 51, 205]. RL methods allow acquiring language through interactions within rich and diverse environments [187], help understanding language acquisition and language pragmatics [154, 22]. "Reward is enough" [257] highlights the necessity of using RL for AI systems to acquire language in its full richness. Indeed, (i) language may be intertwined with other modalities of action and observation, (ii) the utility of language varies according to situations and behaviours, (iii) it is consequential and purposeful, and (iv) some linguistic problems are better solved dynamically, through experience (such as using a diplomatic tone in a speech.) In addition, they allow optimizing a non-differentiable learning signal and thus to handle a more diverse set of objective functions. Finally, they avoid some of the text degeneration issues previously mentioned. So far RL-based text-generation tasks have relied on a pre-training phase to ease learning : the policy language model is trained with SL on the task dataset, before being fine-tuned with policy gradient methods [268] on the task at hand. Those approaches often require human-labelled datasets. Besides, combining pre-training and fine-tuning phases either barely change the policy distribution, or induces *language drift* [156, 185], i.e the generated language drifts semantically or syntactically from natural language.

In this chapter, we aim at learning a conditional language model using *RL from scratch*, so that (i) we get free from datasets with human annotations, and (ii) we avoid the text generation flaws induced by the common methods. While appealing, such an approach requires overcoming the hurdle of the combinatorial language action space, a vocabulary usually containing more than 10,000 words. Yet, while large and discrete, a language action space contains a specific structure, made of all the grammatical, syntactical, and semantics rules of a given language. TrufLL leverages such structure to drive the exploration of the RL-based language agent during training. At each time step of the text generation process, TrufLL truncates its effective action space to a small subset of words provided

by a pretrained task-agnostic language model. Such an approach injects a generic prior linguistic knowledge into the RL algorithm, is usable on tasks lacking in-domain labeled data, and can be easily transferred to new RL-based text generation tasks. Thus, TrufLL can be applied to any language generation task given a generic LM and a reward. We here evaluate it on two Visual Question Generation (VQG) tasks, the synthetic CLEVR dataset [136], and the natural language VQAv2 dataset [111]. Unlike alternative RL from scratch approaches, TrufLL manages to ask meaningful and valid questions, exhibiting success rate and language metrics close to classic pretrain models with labeled data. It also produces original language which differs from the dataset initial distribution, while scaling to large vocabularies containing over 10,000 words.

5.2 Background

Language Generation as an RL Problem. As further described in Section 4.3, we cast the word-based text generation task as a Markov Decision Process to apply RL methods [267]. In this setting, a language model agent generates a sequence of words $w_{<t} = (w_0, w_1, \dots, w_{t-1})$ drawn from a vocabulary $\mathcal{V}$, given an initial context c associated with a reward r_t. Translation, text summarization or image captioning are examples of such tasks respectively using a source sentence, a text article, or an image as a context (c). During this process, the agent may be rewarded with language scores [234], human preferences [261] or task completion scores [262].

Formally, a language generation agent is defined by a policy π_θ (a distribution over $\mathcal{V}$) parametrized by θ, first initialized with the context c. At each time step t, the agent samples a new word w_t from its policy $\pi_\theta(w_t|w_{<t}, c)$, where $s_t = (w_{<t}, c)$ is its current state. It moves to a new state $(w_{<t+1}, c)$ and receives a reward $r_t = r(w_{<t}, c, w_t)$, where $r(.)$ is a reward function relative to the language task. The RL language agent aims to learn a policy that maximizes $\mathbb{E}_{\pi_\theta}[\sum_{t=0}^{T} r_t]$, [2] while generating the sequence of words $w_{<T}$, where $\mathbb{E}_{\pi_\theta}$ is the expectation under π_θ, and T the maximal length of the words sequence.

Policy Gradient This optimization process may be performed through Policy Gradient (PG) algorithms [268]. In the language literature, REINFORCE [286] (further described in Section 4.3) has been used as a simple Monte Carlo approximation of this gradient [262, 169].Yet, in this chapter, we use a Proximal Policy Optimization approach (PPO) [249], known to have a lower variance and better convergence rate ; PPO clips the gradient estimate to have smooth policy updates. For all $0 \leq t \leq T$, let $s_t = (w_{<t}, c)$ and $a_t = w_t$ be the state and action at time t. Policy Gradient methods minimize the objective :

$$L_{PG}(\theta) = \mathbb{E}_{a_t \sim \pi_\theta(.|s_t)} \left[\sum_{t=0}^{T} \log \pi_\theta(a_t|s_t) \hat{A}_t \right] ,$$

2. We cast the language modelling as an episodic problem with $\gamma = 1$ and omit the discount factor in the paper for clarity.

where $\hat{A}_t$ is an estimator of the advantage function, here defined as $\hat{A}_t = r_t - V_\phi(s_t)$ with $V_\phi(s)$ an estimator of the value function $V_\pi(s) = \mathbb{E}_{\pi_\theta}[\sum_t r(s_t, a_t)|s_0 = s]$. PPO then keeps track of the previous policy $\pi_{\theta_{old}}$ before the PG update to compute the training objective :

$$L_{PPO}(\theta) = \mathbb{E}_{a_t \sim \pi_{\theta_{old}}(.|s_t)} \left[\sum_{t=0}^{T} \min(\rho_t^\theta \hat{A}_t, \text{clip}(1-\epsilon, \rho_t^\theta, 1+\epsilon)\hat{A}_t) \right] ,$$

where $\rho_t^\theta = \pi_\theta(a_t|s_t)/\pi_{\theta_{old}}(a_t|s_t)$, ϵ is a hyper-parameter controlling the magnitude of the policy updates, and $\text{clip}(x, a, b)$ is the function that clips x in interval $[a, b]$.

Finally, the training loss is completed first with a value-based loss to learn the baseline V_ϕ that reduces the gradient variance; it computes for each timestep t of an episode the mean squared error $|\sum_{u=t}^{T} r_u - V_\phi(s_t)|^2$ [3]. Secondly, the loss is completed with an entropy term to soften the policy distribution, which computes for each timestep t of an episode $H(\pi_\theta(a_t|s_t))$, where H is the entropy function. As further explained in Section 4.3, the expectation is estimated in practice using a Monte Carlo approach, with an empirical average over a finite batch of trajectories $\tau \in \Upsilon$, i.e a succession of transitions $(s_t, a_t \sim \pi_\theta(.|s_t), r_t)$ from an initial state s_0 to a terminal state s_T. Hence, the final loss L is written as follows :

$$L(\theta, \phi) = \frac{1}{|\Upsilon|} \sum_{\tau \in \Upsilon} \sum_{t=0}^{T} \left\{ \left[\min(\rho_t^\theta \hat{A}_t^\tau, \text{clip}(1-\epsilon, \rho_t^\theta, 1+\epsilon)\hat{A}_t^\tau) + \left| \sum_{u=t}^{T} r_u - V_\phi(s_t) \right|^2 + H(\pi_\theta(a_t|s_t)) \right] \right\} .$$

where $|\Upsilon|$ is the number of trajectories, and $\hat{A}_t^\tau$ is the estimator of the advantage function at timestep t for trajectory τ.

5.3 TrufLL

We here aim at making RL methods feasible in the language setting by dynamically reducing the action space, i.e., by restricting the language agent to select a word within a subset of the vocabulary at each time step. We detail below the action space's truncation model and the associated RL algorithm to learn the language agent.

5.3.1 Dynamic Vocabulary Truncation

TrufLL combines two distinct language models, which share the same vocabulary $\mathcal{V}$: a RL language agent π_θ and a pretrained language model f_{LM}. At each timestep t, TrufLL restricts the vocabulary space of the RL language agent with :

$$\mathcal{V}_t^- = \{w | w \in \mathcal{V}, g_{trunc}(w|w_{<t}) = 1\} ,$$

3. Note that other TD-based losses are applicable [267, 248, 75].

where g_{trunc} is a truncation function based on f_{LM} which either associates 0 or 1 with each word in the vocabulary given the past words $w_{<t}$. From a language modelling perspective, the vocabulary space of the language agent is reduced from $\mathcal{V}$ to $\mathcal{V}^-$ where $|\mathcal{V}^-| \ll |\mathcal{V}|$, with $|\cdot|$ the cardinal of a finite set. From a RL perspective, the RL agent follows a truncated policy π_θ^- which only samples actions over the subset $\mathcal{V}^-$. In practice, such a policy is computed using a masked softmax function over the truncated vocabulary $\mathcal{V}_t^- : \pi_\theta^-(.|w_{<t}, c) = \text{softmax}(m * logits_{\pi_\theta}(w_{<t}, c))$ where $m = 1$ when $g_{trunc}(w|w_{<t}) = 1$ otherwise $m = -\infty$.

5.3.2 Truncation Functions

We here list the different truncation functions g_{trunc} explored through the paper.

Top-k words : This function selects the k words with the highest probability given by $f_{LM}(.|w_{<t})$:

$$g_{\text{top(k)}}(w_t|w_{<t}; k) = \mathbb{1}_{w_t \in \text{top(k)}(f_{LM}(.|w_{<t}))} .$$

Probability threshold (α) : This function only keeps words having a probability $f_{LM}(.|w_{<t})$ greater than α :

$$g_{\text{p}_\text{th}(\alpha)}(w_t|w_{<t}; \alpha) = \mathbb{1}_{f_{LM}(w_t|w_{<t}) > \alpha} .$$

Top-p : This function is based on nucleus sampling [127], and it keeps the most likely words contained in a probability mass p of $f_{LM}(.|w_{<t})$. Formally, we define $\mathcal{V}_t^p$ as :

$$\mathcal{V}_t^p = \underset{|V_t|, V_t \subset \mathcal{V}}{\text{argmin}} \{w|w \in V_t, \sum_{w \in V_t} f_{LM}(w|w_{<t}) > p\} ,$$

and readily, $g_{\text{top(p)}}(w_t|w_{<t}; p) = \mathbb{1}_{w_t \in \mathcal{V}_t^p}$.

Sample (k) : This function randomly samples k words from the language model with replacement to directly build the truncated vocabulary :

$$g_{\text{sample(k)}}(w_t|w_{<t}; k) = \mathbb{1}_{w_t \in \{w_i \sim f_{LM}(.|w_{<t}) \; i \in [\![1,\dots,k]\!]\}} .$$

Only top(k) provides a fixed number of words at each time step. $\text{p}_\text{th}(\alpha)$, top(p), and sample(k) have a dynamic truncation, whose size at t depends on the language model entropy.

5.3.3 Task-Specific vs. Generic LM

We benchmark two types of language models for truncation. On the one hand, we use an *external language model* pretrained on a large task-agnostic language corpora. Such a model provides a generic linguistic prior to the RL agent exploration process, solely encoding syntactic and semantic information. On the other hand, we use a *task-related language model* pretrained on the supervised dataset associated with the task. Such a model provides

a task-specific linguistic prior to the RL language agent, and captures language pragmatics. We emphasize that this chapter aims at leveraging task-agnostic language models as they discard the need for task-specific data. For the sake of completeness, we also study the truncation with the task-related LM as an additional benchmark to assess our approach.

5.4 Experimental Setting

We here list the experimental setting and detail the network and hyperparameters in Appendix A.1.

5.4.1 Visual Question Generation

We showcase TrufLL on the task of Visual Question Generation (VQG) [200], which is a form of Visual Jeopardy ! ™ [85]. There, the language agent observes an image-answer pair and has to generate a question that results in a similar answer, as illustrated in Figure 5.1. Such a task presents multiple advantages. First, by combining vision, scene understanding and language generation, it requires high-level reasoning and exhibits a large spectrum of language difficulties. Secondly, the success criterion is naturally non-differentiable, hence a natural fit for RL methods. Such a criterion, unlike metrics based on ground-truth sentences, allows generating diverse grounded questions given an image-answer pair.

Formally, the initial context c is composed of the image-answer pair $(\mathcal{I}, \mathcal{A})$. The RL agent then generates a sequence of words $w_{<t}$ of maximum length T. We then provide the generated question to a pretrained VQA model. This model takes as inputs the image $\mathcal{I}$, the generated question $w_{<t}$ and outputs a predicted answer $\hat{\mathcal{A}}$. Finally, the agent receives a reward $r(w_t, w_{<t}, c)$ based on $\mathcal{A}$ and $\hat{\mathcal{A}}$.

5.4.2 Datasets

We evaluate TrufLL on the CLEVR and VQAv2 datasets to simulate large-scale VQG datasets. The two datasets have been originally created for the task of Visual Question Answering (VQA), i.e. for multi-modal classification algorithms predicting an answer given an image-question pair.

CLEVR The CLEVR VQA dataset [136] is made of template questions on synthetic images, which contain simple objects with four distinct properties (shape, material, color, size). The vocabulary contains 86 words and 28 potential answers, making it a valuable proof of concept for assessing TrufLL. Both language models are single-layer LSTMs [125] with 512 units, and 512 word embedding dimension. The task-specific LM is trained over the full train dataset of CLEVR questions. The external language model is trained on the mixture of CLOSURE [13] and CLEVR-Dialog [149] datasets. Although those two datasets share the CLEVR vocabulary, their language distribution differs from vanilla CLEVR. Finally, we use a pretrained GT-Vector-NMN [13] to compute the reward

$r(w_t, w_{<t}, c) = \mathbb{1}_{\mathcal{A}=\hat{\mathcal{A}}, t=T-1}$. The Language Model policy is a single-layer LSTM with 64 units. At every time step, the LSTM input is the concatenation of the word embedding of dimension 32, the answer embedding of dimension 32, and the image representation, which is extracted from a pretrained ResNet50 and projected into a tensor of size (32,7,7) before being flattened.

VQAv2 The VQAv2 dataset [111] is made of natural language and open-formed questions on images from the MS-Coco Dataset [175]. It has a vocabulary of 14,810 words and 3,149 answers. The task-specific language model is a one-layer LSTM with 512 units and a 512 word embedding dimension, pretrained over the full training dataset of VQAv2 questions. The External Language Model is Open-AI's GPT-2 [232]. The original language model outputs a probability distribution over $50,257$ tokens, but we use a masked softmax function to restrict the probability distribution to the $14,810$ tokens of the VQAv2 dataset. Unlike most NLP tasks relying on pretrained generic language models, we do not fine-tune it on the task dataset. Instead, we leverage the few-shot generalization capabilities of GPT-2, by feeding the language model with the prompt "Here are a few examples :" followed by 100 random questions $q_{<100}$ from the dataset (outside of the train set). The truncation is then based on the probability distribution $f_{LM}^{gpt2}(.|q_{<100}, w_{<t})$. Finally, we used a pretrained VilBERT to compute the reward [184]. Given the large number of answers, we use as reward a decreasing function of the rank of the reference answer $\mathrm{rk}(\mathcal{A})$: $r(w_t, w_{<t}, c) = \mathbb{1}_{\mathrm{rk}(\mathcal{A})\leq 10, t=T-1} e^{-\mathrm{rk}(\mathcal{A})/2}$, with $\mathrm{rk}(\mathcal{A})$ the rank of the ground-truth answer given by the VQA model, when predicting the actual answer from the terminal state $(c, w_{<T})$. Formally, it is defined as : $\mathrm{rk}(\mathcal{A}) = \mathrm{rank}(\mathrm{VQA}(c, w_{<T})[\mathcal{A}])$, with $\mathrm{VQA}(c, w_{<T})$ the probability distribution given by the VQA model over the set of answers, and rank the function which ranks the probability of answer $\mathcal{A}$ within $\mathrm{VQA}(c, w_{<T})$ probability distribution. The Language Model policy is a single-layer LSTM with 256 units. At every time step, the LSTM input is then the concatenation of the word embedding of dimension 128, the answer embedding of dimension 128, and the image representation, which is the average of 200 bounding box features of dimension 1048, extracted from a faster R-CNN [237].

In these two settings, we acknowledge that the task dataset is still used to train the VQA models. Please note that the VQA modules are only used to model the environment, i.e. to provide a positive/negative feedback to the agent. In other settings, TrufLL would still work if we replace the VQA model by any language interface : text-game (e.g. Zork), expert-systems, or humans. Here, we only use the VQG framework as a proof of concept that natural language can be learned *through pure interaction* given any task reward.

Dataset split For CLEVR (resp. VQAv2), the RL language agent is trained for 50k (resp. 100k) episodes over the first 20k images (resp. all the images) of the training dataset, and is then evaluated on the first 5k (resp. 20k) images of the validation set. For each dataset, we remove yes and no question-answer pairs which frequency largely exceeds other answers, to avoid any bias in the question generation process, as usually done in the VQG litterature [200]. Besides, we uniformly sample the answer in the set of reference answers for each image to reduce

the bias in the distribution of answers. Finally, questions are limited to 20 (resp. 10) words.

Training details We optimize the full loss $L = L_{PPO} + \alpha L_{VF} + \beta L_E$ with $\alpha = 0.5$, $\beta = 0.01$ and a PPO clipping ratio $\epsilon = 0.02$ (resp. 0.01) for CLEVR (resp. VQAv2). We use Adam optimizer [141] with a learning rate (lr) of 10^{-3} for TrufLL and the scratch baseline, 10^{-5} (resp. 10^{-6}) for RL algorithms with a pre-training phase on CLEVR (resp. VQAv2), and $5 * 10^{-4}$ for models including a KL regularization term. We use a batch size (bs) of 128 for all models except the ones with KL regularization, for which we use a batch size of 64. Finally, for the RL from scratch baselines, we perform gradient clipping of 1 (resp. 5) for CLEVR and VQAv2. Further details on hyper-parameters tested are provided in Appendix A.1.

5.4.3 Baselines

In this chapter, we aim to show that a RL language agent can be trained from scratch, i.e. without the usual pre-training phase by solely interacting with another language system, the VQA model, when supported by truncation methods. The truncation with the task-related LM is referred to as TrufLL (Task-LM), while the one with the External LM is referred as TrufLL (Ext-LM). We first emphasize the difficulty of training an RL language agent from scratch through two baselines. We trained a simple on-policy PPO algorithm without any action space pruning, and refer to it as *scratch*. Then, we added a KL regularization term to the loss, $L_{KL} = KL(\pi_\theta || f_{LM})$ and $L' = L + \lambda_{KL} L_{KL}$, to incorporate language prior to the agent as in [132, 133]. We refer to it as *scratch + KL-task* when distilling the task-specific language model, and *scratch + KL-ext* with the external language model. Finally, we include two baselines with a pre-training phase. We trained a language agent on the task-dataset with a log-likelihood objective, and refer to it as *pretrain*. Then, we fine-tune the pretrained language agent with PPO without truncation, and refer to it as *pretrain + RL fine-tune*. These two baselines should be viewed as gold standards as they rely on task-related data.

5.4.4 Metrics and Evaluation Methods

Evaluating text generation is an open-research problem in language literature. We decompose automatic language evaluation into three categories to assess different facets of language, and perform as well a human evaluation study.

Performance metrics. We measure the task-completion score or recall @ 1 which states whether the target answer $\mathcal{A}$ is the top answer of the VQA models, and the recall @ 5 (R@5), which assesses whether $\mathcal{A}$ is in the 5 top answers. These scores measure the task-solving abilities of the agent, but they are also conditioned by the VQA model abilities.

Language Metrics. First, we used n-grams metrics, BLEU [214], METEOR [16] and CIDEr [278], to measure the similarity between the generated question and the reference questions in the evaluation set. The three scores are further described in Section 4.4. While those scores can capture syntactic and semantic properties of language, they

also fall short when dealing with open-form language, e.g. an identical answer may arise from two non-overlapping but syntactically correct questions. Thus, we also compute two metrics assessing the quality of the language independently of reference questions, the perplexity of the question given an external LM (ppl-e), and its perplexity given the task-related LM (ppl-t).

Diversity Metrics. We here estimate a self-BLEU (sBLEU) score [305] over 10 questions generated on the same image-answer pair, described in Section 4.4. Although such score detects potential mode collapse, i.e., when the language utters identical sequences of words, it also values babbling, i.e., outputting random words. We thus also measure the probability mass of the ten most frequent words [47], and refer to it as peakiness (peak).

Human Evaluation. On the VQAv2 task, we also performed human evaluation by surveying 53 participants on the first 50 questions produced by some of the models at test time. The study is based on pairwise comparison of question samples produced by the concurrent algorithms according to four criteria. First, we evaluated the language quality of the question samples, by asking the participants to select the most syntactically and semantically correct question among the two samples of the questions pair. Secondly, we evaluated language grounding, i.e adequacy of the sample to the image-answer pair, by asking the participants to select the question most suitable given the two elements. Thirdly, we evaluated the language originality and diversity, by asking participants to select the question the most different from the dataset reference question. Finally, we evaluated the number of syntax errors by asking participants to tick the question if it is grammatically incorrect. Examples of questions asked during the study are included in the Appendix A.3.

5.4.5 Sampling methods for text generation

When generating text from a trained language model, the quality and diversity of samples depend on the decoding algorithm [301]. We consider three text generation methods. *greedy* uses the argmax of the policy, while *sampling* uses the multinomial distribution. Finally, we sampled ten text sequences from the policy, and selected the one with the lowest perplexity according to the external language model, and refer to it as *lm-ranking*.

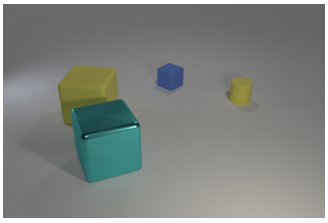
5.5 Results

5.5.1 CLEVR results

Quantitative performance : In Table 5.1, vanilla RL from scratch fails to have a decent performance even with synthetic language. Besides, adding a KL regularisation term does kick-start the learning process. Yet, as soon as we apply the dynamic truncation, TrufLL matches the pretrained baselines performance when using the external LM, and even outperforms them with the task-specific LM. In this synthetic VQG setting, TrufLL seems to be a viable and promising procedure to learn a RL language agent from scratch. Pretrained baselines have high language

Method	Score	R@5	BLEU	Meteor	CIDEr	ppl-t (↓)	ppl-e (↓)	sBLEU (↓)	peak.(↓)
Pretrain	0.30	0.71	0.19	0.38	0.83	3.1	31	0.44	0.96
Pretrain + RL fine-tune	0.44	0.86	0.17	0.34	0.70	4.0	35	0.46	0.95
Scratch	0.17	0.47	0.05	0.08	0.10	10^9	10^6	**0.14**	**0.26**
Scratch + KL-task	0.14	0.38	0.15	0.30	0.53	92	10^2	0.34	0.94
Scratch + KL-ext	0.17	0.44	0.14	0.27	0.43	10^4	28	0.37	0.95
TrufLL (Task-LM)	**0.56**	0.90	**0.17**	**0.32**	**0.66**	**3.4**	23	0.95	1.00
TrufLL (Ext-LM)	0.48	**0.93**	0.08	0.18	0.34($\pm$0.10)	10^3	**3.0**	0.95	1.00

TABLE 5.1 – **CLEVR metrics** on 5k test episodes while training an agent on 20k Images over 50k train episodes. Scores are averaged over the three decoding procedures mentioned in Section 5.4.5 and over 5 seeds ; standard deviations are displayed when greater than 0.01 for accuracy metrics. We here report the models with the highest task-success :, i.e. the *scratch+KL* baselines with $\lambda_{KL} = 0.1$, and the truncation model with a probability threshold, $\mathrm{p_{th}}(\alpha = 0.05)$. Overall best values are underlined, best values without task-data (from scratch) are in bold.



Human	There is a blue thing that is the same shape as the big cyan metallic object ; what is its size ? **A :Small**
pretrain	There is a red metallic object that is the same size as the yellow rubber block ; what is its size ?
pretrain + RL	What size is the thing that is the same color as the matte cube ? ☑
scratch	size sphere small blue or a yellow green large else in cylinders cubes color and how matte objects cube
scratch+KL-task	How big is the shiny cylinder ?
scratch+KL-ext	How many other objects in the are of same color as that shiny object ?
TrufLL (Task-LM)	How big is the thing that is to the right of the big matte thing ? ☑
TrufLL (Ext-LM)	What is the size of the thing that is right of the big cyan thing and is the same shape ? ☑

Human	What color is the cat **A :Black**
pretrain	What color is the cat's collar ? ☑
pretrain + RL	What color is the cat ? ☑
scratch	AmazingAmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What color is their hat of the fingers of this ?
scratch+KL-ext	The the first time is a bit of the way
TrufLL (Task-LM)	What color is her outfit ? ☑
TrufLL (Ext-LM)	What color can these cats look like in real life ? ☑

FIGURE 5.2 – Samples on CLEVR and VQA : the checkbox indicates that the question generates the correct answer.

scores when assessed with dataset-based metrics, e.g BLEU or task-perplexity. Yet, they also remain close to the original dataset distribution with a medium external perplexity. Noticeably, TrufLL with the task-specific LM follows the same pattern. On the other hand, TrufLL with the external LM reports poor dataset-based language scores, while maintaining a low external perplexity. Therefore, TrufLL seems to correctly capture the language distribution of the initial LM. As the performance score is high when using an external LM, it suggests that our approach can learn a policy on a language task without the need of a task-related dataset. Less positively, TrufLL diversity metrics suggest potential mode collapse, with a high peakiness and self-BLEU score.

Qualitative performance : We display qualitative samples In Figure 5.2 and Appendix A.4. On the one hand, the pretrained baselines generate either a question inconsistent with the visual context, or which fails to answer the expected answer. They inaccurately capture the pragmatics of the task. On the other hand, TrufLL generate adequate questions, resulting in the expected answer. Interestingly, they are often grounded with different objects of the image. It is remarkable that TrufLL with a generic LM still manages to capture the necessary subtleties of VQG, without any prior task knowledge. Despite a peaky distribution, TrufLL has moderate repetitions across images, and is mostly overconfident. As for the scratch+KL samples, they are either not grounded, or showcase degenerated language.

Trunc.	Score	BLEU	CIDEr	ppl-e(↓)	sBLEU(↓)
TrufLL (Task-LM)					
$\mathrm{top(k)}$	0.50	0.12	0.32	100	0.93
$\mathrm{p_{th}}(\alpha)$	**0.54**	0.17	0.65	24	0.95
$\mathrm{top(p)}$	0.51	0.17	0.69	**12**	0.96
$\mathrm{sample(k)}$	0.50	**0.18**	**0.73**	16	**0.89**
TrufLL (Ext-LM)					
$\mathrm{top(k)}$	**0.52**	0.06	0.15	151	0.94
$\mathrm{p_{th}}(\alpha)$	0.48	0.08	0.34($\pm$0.10)	3.0	0.95
$\mathrm{top(p)}$	0.45	0.10	0.40($\pm$0.17)	3.3	**0.92**
$\mathrm{sample(k)}$	0.41	**0.13**	**0.46**($\pm$0.16)	**2.7**	**0.92**

TABLE 5.2 – **CLEVR task :** Truncation functions with parameters : $\mathrm{top(k=10)}$, $\mathrm{p_{th}}(\alpha = 0.05)$ $\mathrm{top(p=0.85)}$, $\mathrm{sample(k=20)}$. Overall best values are underlined, best values for each TrufLL algorithms are in bold.

Method	Score	R@5	BLEU	Meteor	CIDEr	ppl-t (↓)	ppl-e (↓)	sBLEU (↓)	peak.(↓)
Pretrain	0.38	0.59	0.30	0.40	0.93	12	24	0.80	0.99
Pretrain + RL fine-tune	0.41	0.63	0.31	0.41	0.98	21	50	0.78	0.99
Scratch	0.01	0.04	0.00	0.00	0.00	10^7	10^6	0.75	1.00
Scratch + KL-task	0.11	0.29	**0.24**	**0.27**	**0.24**	10^2	10^2	0.27	0.74
Scratch + KL-ext	0.01	0.05	0.06	0.04	0.01	10^6	10^3	**0.10**	**0.20**
TrufLL (Task-LM)	**0.35**	**0.56**	0.21	0.15	0.11	**24**	10^2	0.78	0.99
TrufLL (Ext-LM)	0.34	0.52	0.18	0.15	0.04	10^2	**24**	0.83	0.99

TABLE 5.3 – **VQAv2** metrics on 20k test episodes with 100k train episodes. Scores are averaged over the three decoding procedures. We report the models with the highest task-success, i.e. *scratch+KL* with $\lambda_{KL} = 0.05$, and truncation with a probability threshold, $\mathrm{p_{th}}(\alpha = 0.005)$ for TrufLL with (Task-LM) and $\mathrm{p_{th}}(\alpha = 0.0075)$ for (Ext-LM). Overall best values are underlined, best values without task-data are in bold.

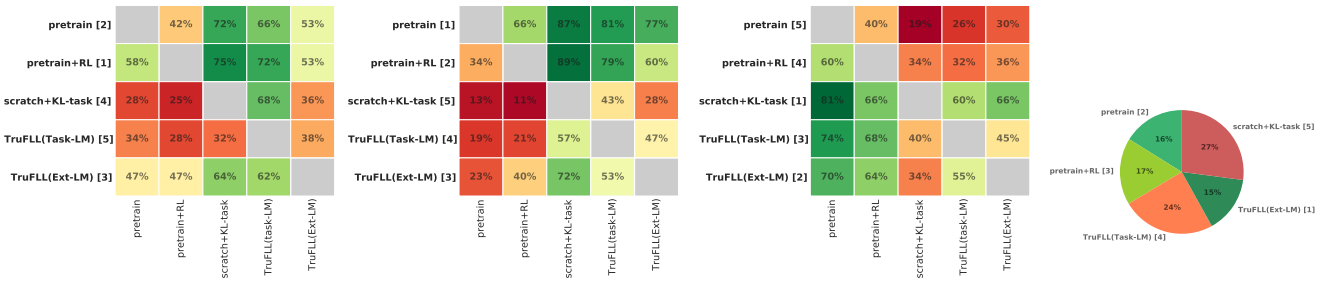


FIGURE 5.3 – **VQAv2** results for Human Evaluation study detailed in Section 5.4.4. From left to right : evaluation of (a) language quality, (b) language grounding, (c) language diversity/originality, (d) syntax errors. Matrices (a),(b) and (c) are pairwise comparisons : each cell displays the proportion of questions chosen for the models in the row (bold) when compared to the concurrent model in the column. Figure (d) displays the proportion of incorrect questions coming from each model among all incorrect samples. In all figures, bracket numbers indicates the model rank per criteria, from 1="best" to 5="worst".

Truncation function in CLEVR : In Table 5.2, we evaluate the different truncation functions defined in Section 5.3. While all truncation methods report similar task performance, the dynamic truncation functions, i.e. $\mathrm{p_{th}}(\alpha)$, $\mathrm{top(p)}$ and $\mathrm{sample(k)}$, outperform the $\mathrm{top(k)}$ regarding language metrics. Interestingly, the $\mathrm{sample(k)}$ one, which generates a stochastic truncated action space, while having a lower performance, yields to the most correct and diverse language, with higher language scores and a lower self-BLEU. A stochastic action space might be harder to explore efficiently for reaching good task-solving abilities, but might strengthen the agent language generation properties.

5.5.2 VQAv2 task

In CLEVR, we observe that TrufLL seems a promising approach to learn a language policy from scratch by solely interacting with another language system. We scale our approach to natural language with large vocabulary (15k

tokens) through the VQAv2 dataset.

Quantitative performance : Table 5.3 reports the VQAv2 results, for which TrufLL and the baselines present a similar trend than on CLEVR. First, the scratch baselines keep failing to learn a valuable policy, with performance scores and n-grams metrics close to zero. Although TrufLL does not outperform the performance of the pretrained baselines anymore, it still leads to similar performances, and satisfactory language scores. The similarity between TrufLL (Task-LM) and TrufLL (Ext-LM) results suggests that the truncation approach is viable when using a generic LM whose original vocabulary distribution differs from the task. Interestingly, TrufLL displays a self-BLEU score similar to the pretrained baselines. This suggests that the poor diversity behavior observed on CLEVR is likely attributable to the small vocabulary and synthetic language distribution.

Qualitative performance : In Figure 5.2 and Appendix A.4, we display question samples for all models. TrufLL and the pretrained baselines successfully generate a question giving the expected answer ("Black"), while the RL from scratch baselines fail, and even showcase degenerated language. Pretrained baselines tend to output a question closer to the reference question whereas TrufLL outputs original questions which differs from the VQA distribution, yet consistent with the context.

Human Evaluation : Figure 5.3 details the Human Evaluation results. Among the RL from scratch baselines, we selected scratch+KL-task as the only model producing sometimes meaningful questions. Yet, it fails to generate correct and grounded language ; it is thus not a viable approach despite its diverse output. In line with the automatic metrics, the supervised baselines produce the best language, while being accurately grounded. Yet, they exhibit significantly less diversity with the reference language ; this suggests in particular that pretrain+RL fails to go beyond the initial task-data distribution. Finally, unlike TrufLL (Task-LM) which suffers from syntactic errors, TrufLL (Ext-LM) produces language that qualitatively competes with pretrain models (53%), with a similar ratio of syntactic uncorrect samples. Although its questions are less grounded, they are diverse, which suggests that they follow a different distribution from the initial VQA dataset. It confirms that TrufLL (Ext-LM) could be an alternative approach as it has an excellent trade-off between language quality, diversity, and grounding.

Decoding procedure : In Table 5.4, we evaluate the text sampling procedures described in Section 5.4.5. While greedy decoding produces the best outcome for pretrained models, lm-ranking provides an excellent trade-off between task performance and language quality with RL-based methods. As PG solely optimizes the task success ratio, this may reduce overall language quality, the re-ranking thus retrieves the best syntactically sentences a posteriori.

5.5.3 Discussion

Removing the truncation at evaluation with off-policy RL. So far, TrufLL directly learns the truncated policy over the truncated vocabulary $\mathcal{V}_t^-$ in an on-policy scheme. Hence, the algorithm requires the truncation, and a fortiori the language model, at test time. In this section, we investigate if we can directly learn a policy over the full vocabulary,

Method	Text-gen	Score	BLEU	CIDEr	ppl-e
pretrain	greedy	0.40	0.32	1.01	51
	sampling	0.37	0.30	0.88	62
	lm-ranking	0.37	0.14	0.87	54
pretrain + RL	greedy	0.42	0.32	1.05	55
	sampling	0.40	0.30	0.92	71
	lm-ranking	0.40	0.31	0.99	26
TrufLL (Task-LM)	greedy	**0.36**	0.20	0.11	366
	sampling	0.35	0.20	0.11	337
	lm-ranking	0.34	**0.21**	0.11	95
TrufLL (Ext-LM)	greedy	**0.36**	0.18	0.04	25
	sampling	0.34	0.18	0.04	28
	lm-ranking	0.33	0.19	**0.15**	**20**

TABLE 5.4 – **VQAv2 :** Ablation on the sampling methods. Overall best values are underlined, TrufLL best values are in bold.

and thus removing the truncation at test time. In such a setting, we adopt an off-policy training scheme, where the trajectories used to learn the behavior π_θ at training time are sampled under a different policy, the truncated policy $\pi_{\bar{\theta}}$. Thus, we need to unbiased the PG by using an importance sampling term between the exploratory policy $\pi_{\bar{\theta}}$ and the behavior policy π_θ [57]. Formally, the off-policy PPO loss is defined by :

$$L_{PPO}^{off} = \mathbb{E}_{\pi_{\bar{\theta}}}\big[\min(\bar{\rho}_t A_t, \text{clip}(1-\epsilon, \bar{\rho}_t, 1+\epsilon)A_t)\big],$$

where $\bar{\rho} = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} * \frac{\pi_{\theta_{old}}(a_t|s_t)}{\pi_{\bar{\theta}_{old}}(a_t|s_t)}$ is the new ratio. [4]

Table 5.5 displays the on-policy and off-policy results on both VQG tasks for TrufLL (task-LM), and is further detailed in Appendix A.2.3. We also monitor the probability mass of the policy attributed to the truncated action space (sumVA). The policy only samples words within the truncated action space when $\text{sumVA} = 1$, without needing the truncation. On CLEVR, the $\text{TrufLL}_{\text{off}}$ has lower - yet close - performance on language and task scores than TrufLL. As its sumVA ratios are very close to 1, the agent has learned to generalize over the full vocabulary. However, the approach does not manage to sufficiently scale to VQAv2. It could be improved with regularisation techniques and the use of TruFLL within state-of-the-art off-policy RL algorithms. We leave such possibilities to future works.

Algo	Score	BLEU	CIDEr	ppl-e	sBLEU	sumVA
CLEVR						
TrufLL	0.56	0.17	0.06	10^3	0.78	N.A
$\text{TrufLL}_{\text{off}}$	0.50	0.14	0.43	10^4	0.88	0.96
VQAv2						
TrufLL	0.35	0.21	0.11	10^4	0.36	N.A
$\text{TrufLL}_{\text{off}}$	0.07	0.03	0.01	10^4	0.05	0.08

TABLE 5.5 – On-policy vs. off-policy scores : when training with an off-policy loss, we remove the truncation at test time.

Additional experiments. We sweep over truncation hyper-parameters in Table A.1 of Appendix A.2. In Table A.3, we observe that rewarding an agent with a BLEU score is sub-optimal in both language and task scores on CLEVR. In VQA, we apply temperature scheduling on the LM to perform fine-grained truncations in Table A.4 of A.2.2. Finally, we explore TrufLL with a pre-training phase in Table A.5.

4. Note that we did not simplify the expression to highlight the importance sampling ratio.

5.6 Related work

Reinforcement Learning and NLP Tasks. Following [258, 162], recent RL-based task-oriented dialogues [55, 51, 164, 205] have been developed. There, the agent is generally pretrained with a SL phase followed by a RL fine-tuning phase to learn the policy's language model. Yang et al. [295], Fan et al. [77] focused on tackling VQG tasks with RL, respectively on CLEVR and on the VQG dataset. Yet, the former uses slot filling with template questions, while the later computes a mixed objective with a MLE loss using ground-truth sentences. Bahdanau et al. [12], Rennie et al. [238] use RL to train language models as an alternative to SL to prevent typical text degeneration issues, but within training algorithms relying on ground-truth examples from labelled datasets.

RL methods for Language Action Spaces. Several RL algorithms have been developed to tackle large discrete action spaces. Hence, Dulac-Arnold et al. [73], Tennenholtz and Mannor [273], Chandak et al. [43] embed the actions into a continuous action space, and then use classic RL algorithms to learn a policy over this continuous space. Zahavy et al. [300], Seurin et al. [254] proposes Q-learning algorithms with an elimination signal to eliminate forbidden actions. Closer to our work, a few algorithms [5] use the structure of language to prune the action space of text-based games, but within value-based algorithms, which are less scalable to large vocabularies. Similarly to TrufLL, CALM [296] combines a pretrained language model to prune the action space with a Deep-Q network, aka DRNN [119]. Yet, its truncation language model remains fine-tuned on the RL dataset. Besides, CALM is only evaluated on a vocabulary of 697 tokens, and on 4-words action sequences.

Learning Language Models from scratch. [308, 95] finetune pretrained GPT-2 models with RL for language generation tasks without task-related data, only using reward signals. Yet, they still face optimization and computational challenges [215].

5.7 Conclusion

We proposed TrufLL, an original approach to learn a natural language generation (NLG) task from scratch using RL. To our knowledge, this is the first RL-based algorithm dedicated to learning a word-based text-generation task, which does not rely on a pre-training phase while scaling to large vocabularies. Although it comes with its limitations, the truncated RL algorithm provided by TrufLL gets free from labelled data in task-oriented language models, presents interesting language generation properties, and provides a generic and transferable method to learn any NLG problem.

One limitation of TrufLL is its tendency to suffer from "mode collapse", i.e in the context of Visual Question Generation, the RL Language agent repeats the same "safe" question for multiple images. In the next chapter, we develop a Deep Generative Model for sequential data with stochastic latent states : such a model, by better taking

in account the variability in the text input data, could mitigate the "mode collapse" issue in RL-based Language Models.

Chapitre 6

The Monte Carlo Transformer : A Stochastic Self-attention Model for Sequence Prediction

This chapter introduces the Sequential Monte Carlo Transformer, an original latent data model that naturally captures the observations distribution using a transformer architecture. The keys, queries, values and attention vectors of the network are considered as the unobserved stochastic states of its hidden structure. This generative model is such that at each time step the received observation is a random function of these past states. In this general state-space setting, we use Sequential Monte Carlo methods to approximate the posterior distribution of the states given the observations, and to estimate the gradient of the log-likelihood. Such approaches allow to capture the predictive distribution of new observations instead of providing single-point estimates. Our model correctly predicts the data distribution in three synthetic datasets as opposed to other Bayesian recurrent neural networks, and it provides consistent uncertainty estimates when applied to five real-world time-series datasets.

6.1 Introduction

Many critical applications (e.g. medical diagnosis or autonomous driving) require accurate forecasts while detecting unreliable predictions, that may arise from anomalies, missing information, or unknown situations. While neural networks excel at predictive tasks, they often solely output a single-point estimate, lacking uncertainty measures to assess their confidence about their predictions. To overcome this limitation, an open research question is the design of neural generative models able to output a predictive distribution instead of single point-estimates. First, such distribution would naturally provide the desired uncertainty measures over the model predictions. Second, learning

algorithms can build upon such uncertainty measurements to improve their predictive performance such as active learning [36] or exploration in reinforcement learning [99, 89]. Third, they may better model incoming sources of variability, e.g. observation noise, missing information, or model misspecification.

On the one hand, Bayesian statistics offer a mathematically grounded framework to reason about uncertainty, and has long been extended to neural networks [206, 190]. Among recent methods, Bayesian neural networks (BNNs) estimate a posterior distribution of the target given the input variables by injecting stochasticity in the network parameters [27, 49] or casting dropout as a variational predictive distribution [94]. However, such models tend to be overconfident, leading to poorly calibrated uncertainty estimates [87]. On the other hand, concurrent frequentist approaches have been developed to overcome the computational burden of BNNs, by either computing ensembling networks [128, 130, 250] or directly optimizing uncertainty metrics [220]. Yet, such methods suffer their own pitfalls [8]. Uncertainty estimation have been less explored in sequential prediction problems. Only a few techniques have been adapted to recurrent neural networks [88, 306] or transformer networks [250]. Specifically, sequential transformers were explored on single-point sequence prediction with quantile regressions [259, 171, 289], and it remains an open-problem to train sequential transformers that output a complete predictive distribution.

To that end, we present the Sequential Monte Carlo (SMC) recurrent Transformer, which models uncertainty by introducing stochastic hidden states in the network architecture, as in [49]. Specifically, we cast the transformer self-attention parameters as unobserved latent states evolving randomly through time. The model relies on a dynamical system, capturing the uncertainty by replacing deterministic self-attention sequences with latent trajectories. However, the introduction of unobserved stochastic variables in the neural architecture makes the log-likelihood of the observations intractable, requiring approximation techniques in the training algorithm.

In this chapter, we propose to use particle filtering and smoothing methods to draw samples from the distribution of hidden states given observations. Standard implementations of Sequential Monte Carlo methods are based on the auxiliary particle filter [178, 224], which is a generalization of [109, 145] and are theoretically grounded by numerous works in the context of Hidden Markov Models [58, 39, 59, 72, 212].

Fitting the Transformer approach to general state space modeling provides a new promising and interpretable statistical framework for sequential data and recurrent neural networks. From a statistical perspective, the SMC Transformer provides an efficient way of writing each observation as a mixture of previous data, while the approximated posterior distribution of the unobserved states captures the states dynamics. From a practical perspective, the SMC Transformer requires extra-computation at training time, but only needs a single forward pass at evaluation as opposed for example to MC dropout methods [94].

We evaluate the SMC Transformer model on three synthetic datasets and five real-world time-series forecasting tasks. We show that the SMC Transformer manages to capture the known observation models in the synthetic setting, and outperforms all concurrent baselines when measuring classic predictive intervals metrics in the real-world setting. Finally, we apply the SMC Transformer on the task of Language Modelling, where we showcase its

ability to foster diversity in text generation, while also pointing out the limits of the model when applied on complex NLG problems.

6.2 Background

6.2.1 Sequential Monte Carlo Methods

In real-world machine learning applications, the latent states of parametric models and the observations tend to be noisy. Generative models have thus been used to replace deterministic states with unobserved random variables.

However, this leads to an intractable log-likelihood function of the observed data $Y_{0:T}$, where for any sequence $(u_k)_{k\geqslant 0}$ and any $s \leqslant t$, $u_{s:t}$ is a short-hand notation for $(u_s, \dots, u_t)$. Indeed, this quantity is obtained by integrating out all latent variables, which cannot be done analytically. Fortunately, a gradient descent algorithm may still be defined using Fisher's identity to estimate the score function [39] :

$$\nabla_\theta \log p_\theta(Y_{0:T}) = \mathbb{E}_\theta[\nabla_\theta \log p_\theta(X_{0:T}, Y_{0:T}) | Y_{0:T}] \,, \tag{6.1}$$

where θ denotes the unknown parameters of the model, $X_{0:T}$ denotes all the unobserved states, p_θ the joint probability distribution of the observations $Y_{0:T}$, and the latent states and $\mathbb{E}_\theta$ the expectation under p_θ.

Sequential Monte Carlo methods, also called particle filtering and smoothing algorithms, aim at approximating the conditional distributions of the hidden states given the observations by a set of random samples associated with non-negative importance weights. These algorithms - described in details in Section 3.3.1 - combine two steps : (i) a sequential importance sampling step, and (ii) an importance resampling step which selects particles according to their importance weights. Following Eq (6.1), $\nabla_\theta \log p_\theta(Y_{0:T})$ is then approximated by a weighted sample mean of the form

$$S^M_{\theta,T} = \sum_{m=1}^{M} \omega_T^m \nabla_\theta \log p_\theta(\xi^m_{0:T}, Y_{0:T}) \,, \tag{6.2}$$

where $(\omega_T^m)_{1\leqslant m\leqslant M}$ are such that $\sum_{m=1}^M \omega_T^m = 1$ and where $\xi^m_{0:T}$ are trajectories approximately sampled from the posterior distribution of $X_{0:T}$ given $Y_{0:T}$ parametrized by θ. Such an approximation of the objective function can be plugged into any stochastic gradient algorithm to find a local minimum of the negative log-likelihood function $\theta \mapsto -\log p_\theta(Y_{0:T})$.

6.2.2 The Transformer model

Transformers are neural networks developed as an alternative to recurrent and convolution layers for sequence modeling [277]. They rely entirely on (self)-attention mechanisms [11, 176] to model global dependencies regardless of their distance in input or output sequences. In their original forms, transformers are sequence-to-sequence models

with an encoder module to process the input, and a decoder module to generate sequence of tokens. In sequential data problems (e.g. time-series forecasting), a common practice is to only keep the decoder [180, 231].

Formally, given the sequence of multivariate observations $(Y_s)_{0\leqslant s\leqslant t}$ in $\mathbb{R}^p$, we build an auto-regressive model that predicts an output Y_{t+1} from the past input data $Y_{0:t}$. In transformer networks, the self-attention modules first associate each input data Y_s with a query q_s and a set of key-value (κ_s, v_s), where the queries, keys and values are linear projections of the input :

$$q_s = Y_s W^q \;, \quad \kappa_s = Y_s W^\kappa \;, \quad v_s = Y_s W^v \;,$$

where Y_s is a row vector and W^q, W^κ and W^v are unknown weight matrices in $\mathbb{R}^{p\times \mathsf{d}}$. A self-attention score is computed from a dot-product of queries and keys to determine how much focus to place on each input in Y_s to predict Y_{t+1}. Then, the output attention vector z_t is the linear combination of all values weighted with each attention score :

$$z_t = \mathrm{softmax}(q_t K_t^\top / \sqrt{\mathsf{d}}) \bullet V_t \;, \tag{6.3}$$

where K_t (resp. V_t) is a matrix whose rows are $(\kappa_s)_{0\leqslant s\leqslant t}$ (resp. $(v_s)_{0\leqslant s\leqslant t}$), d is the dimension of the key, and $\bullet$ is the dot product.

Transformers generally rely on *multi-head* self-attention, where the input data is independently processed by H self-attention modules. This leads to H outputs, then concatenated back together to form the final attention output vector. Further details on the model are provided in Section 3.2.2.

6.3 The SMC Transformer

6.3.1 Generative model with stochastic self-attention

In this section, we introduce the SMC Transformer, a recurrent generative neural network for sequential data based on a stochastic self-attention model. When processing sequentially elements $(Y_s)_{0\leqslant s\leqslant t-1}$ to predict Y_t, we define the (key, queries, values) of the *stochastic* self-attention layer of the SMC Transformer as follows :

$$q_s = Y_{s-1} W^q + \varepsilon_s^q \;, \quad \kappa_s = Y_{s-1} W^\kappa + \varepsilon_s^\kappa \;, \quad v_s = Y_{s-1} W^v + \varepsilon_s^v \;, \tag{6.4}$$

where $(\varepsilon_s^q, \varepsilon_s^\kappa, \varepsilon_s^v)$ are independent centered Gaussian random vectors in $\mathbb{R}^{\mathsf{d}}$ with unknown covariance matrices $(\Sigma^q, \Sigma^\kappa, \Sigma^v)$. As in Eq 6.3, we then compute the stochastic self-attention vector at time t :

$$z_t = \mathrm{softmax}\left(q_t^\top K_t / \sqrt{\mathsf{d}}\right) \bullet V_t + \varepsilon_t^z \;, \tag{6.5}$$

where K_t (resp. V_t) is a matrix whose rows are $(\kappa_s)_{0\leq s\leq t}$ (resp. $(v_s)_{0\leq s\leq t}$), d is the dimension of the key, $\cdot$ is the dot product, and ε^z is a centered Gaussian random vector in $\mathbb{R}^{\mathrm{d}}$ with unknown covariance matrix Σ^z, independent of $(\varepsilon^q_s, \varepsilon^\kappa_s, \varepsilon^v_s)$. Finally, in a regression framework, the observation model is given by :

$$Y_t = G_\eta(z_t) + \varepsilon^Y_t \ ,$$

where G_η is a feedforward neural network parametrized by η (detailed in Section 3.2.2 and referred there as the Transformer residual layers), and ε^Y_t is a centered noise such as a centered Gaussian random vector with unknown variance Σ^{Y}. This one-layer SMC Transformer can be extended to a multi-layer model with $L > 1$ layers. We first encode the input data $Y_{0:T}$ with $L-1$ bottom layers of a classical Transformer decoder, and then use this encoding as the input of the upper stochastic self-attention layer.

By injecting noise in the self-attention parameters of the transformer model, we propose a recurrent generative neural network to predict the conditional distribution of Y_{t+1} given past observations $Y_{0:t}$. The next section presents the training algorithm to learn the unknown parameters

$$\theta = (\eta, W^q, W^\kappa, W^v, \Sigma^Y, \Sigma^q, \Sigma^\kappa, \Sigma^v)$$

of this network.

The model given in (6.5) does not enjoy forgetting properties as the self-attention vector z_t depends on the complete trajectories of keys and values through K_t and V_t. Yet, a straightforward fixed-lag model can be derived from (6.5), so that the proposed dynamical system is computed only on a fixed number of previous keys and values in K_t and V_t. Such a model allows to simultaneously tune the desired forgetting property and the practical performance of the model.

6.3.2 Training Procedure

Gradient Estimation By section 6.3.1, the unobserved state at time t is $X_t = (z_t, q_t, \kappa_t, v_t)$, which depends on $X_{0:t-1}$ and the current observation Y_{t-1}. The next observation Y_t depends on the current state X_t. Figure 6.1 proposes a graphical representation of the state-space model, which describes the dependencies between the latent unobserved states (estimated as a set of M particles), the observations, and the outputs.

Lemma 1. *For all θ, the gradient of the log-likelihood is given by :*

$$\nabla_\theta \log p_\theta(Y_{0:T}) = \sum_{t=0}^{T} \mathbb{E}_\theta \left[\{ \nabla_\theta \log p_\theta(X_t | X_{0:t-1}, Y_{t-1}) + \nabla_\theta \log p_\theta(Y_t | X_t) \} | Y_{0:T} \right] \ ,$$

with the convention $\nabla_\theta \log p_\theta(X_0 | X_{0:-1}, Y_{-1}) = \nabla_\theta \log p_\theta(X_0)$.

Proof. For all $0 \leq t \leq T$, the unobserved state at time t is $X_t = (z_t, q_t, \kappa_t, v_t)$. Conditionally on $(X_{0:t-1}, Y_{t-1})$, (q_t, κ_t, v_t) depends on Y_{t-1} only and, by (6.5), z_t depends only on (q_t, κ_t, v_t) and on and $X_{1:t-1}$. On the other hand, conditionally on $(X_{0:t}, Y_{t-1})$, Y_t depends on z_t (i.e. X_t) only. The complete-data likelihood after T timesteps may therefore be written :

$$p_\theta(Y_{0:T}, X_{0:T}) = \prod_{t=0}^{T} p_\theta(X_t|X_{0:t-1}, Y_{t-1}) p_\theta(Y_t|X_t) \ ,$$

with the convention $p_\theta(X_1|X_{0:-1}, Y_{-1}) = p_\theta(X_0)$. By (6.1),

$$\begin{aligned} \nabla_\theta \log p_\theta(Y_{0:T}) &= \mathbb{E}_\theta[\nabla_\theta \log p_\theta(X_{0:T}, Y_{0:T})|Y_{0:T}] \ , \\ &= \mathbb{E}_\theta \left[\sum_{t=0}^{T} \{\nabla_\theta \log p_\theta(X_t|X_{0:t-1}, Y_{t-1}) + \nabla_\theta \log p_\theta(Y_t|X_t)\} \middle| Y_{0:T} \right] \ . \end{aligned}$$

□

The associated observation model in the regression setting is $p_\theta(Y_t|X_t) = \varphi_{G_\eta(z_t),\Sigma^Y}(Y_t)$, where $\varphi_{\mu,\Sigma}$ is the Gaussian probability density function with mean μ and covariance matrix Σ. Then, the sequential Monte Carlo algorithm approximates $\nabla_\theta \log p_\theta(Y_{1:T})$ by a weighted sample mean :

$$S^M_{\theta,T} = \sum_{m=1}^{M} \omega_T^m \sum_{t=0}^{T} \left[\nabla_\theta \log p_\theta(\xi_t^m|\xi_{1:t-1}^m, Y_{t-1}) + \nabla_\theta \log p_\theta(Y_t|\xi_t^m) \right] \ , \tag{6.6}$$

where $(\omega_T^m)_{1 \leqslant m \leqslant M}$ are the importance weights and $\xi_{0:T}^m = (z_{0:T}^m, q_{0:T}^m, \kappa_{0:T}^m, v_{0:T}^m)$ are the trajectories sampled from the posterior distribution of $X_{0:T}$ given $Y_{0:T}$. In practice, we sample both the importance weights and the trajectories by using the particle smoother described in Algorithm 2. Thanks to Fisher's identity, this approximation only requires to compute the gradient of state model $\theta \mapsto \log p_\theta(\xi_t^m|\xi_{0:t-1}^m, Y_{t-1})$ and the gradient of the observation model $\theta \mapsto \log p_\theta(Y_t|\xi_t^m)$. There is no need to compute the gradient of the weights ω_T^m which depend on the parameter θ. The loss function used to train the model is therefore

$$\theta \mapsto - \sum_{m=1}^{M} \omega_T^m \sum_{t=0}^{T} \left[\log p_\theta(\xi_t^m|\xi_{1:t-1}^m, Y_{t-1}) + \log p_\theta(Y_t|\xi_t^m) \right] \ .$$

Model Update In this chapter, we propose to estimate all the parameters of the recurrent architecture based on a gradient descent using $S^M_{\theta,T}$. All parameters related to the noise (the covariance matrices) are estimated using an explicit Expectation Maximization (EM) update [62] each time a batch of observations is processed. For each sequence of observations, the EM update relies on the approximation of the intermediate quantity

$$\mathbb{E}[\log p_\theta(Y_{0:T}, X_{0:T})|Y_{0:T}] = \sum_{t=0}^{T} \mathbb{E}[\log p_\theta(X_t|X_{0:t-1}, Y_{t-1}) + \log p_\theta(Y_t|X_t)|Y_{0:T}]$$

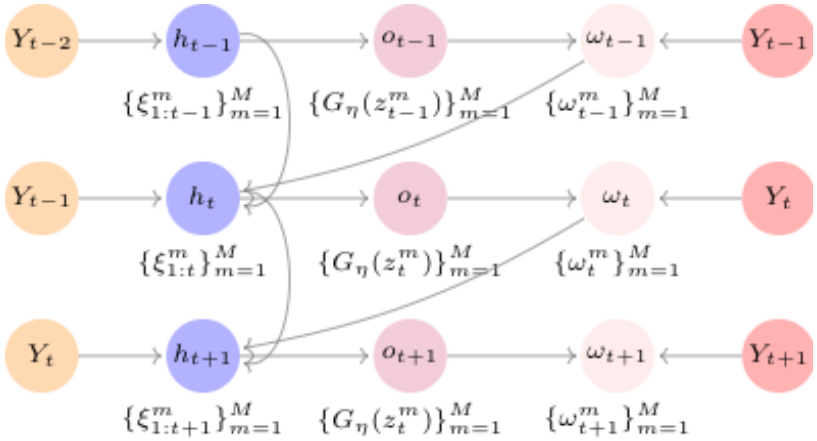


FIGURE 6.1 – Graphical illustration of the SMC Transformer seen as a recurrent neural network. At each time step t, the dynamical systems takes as input the current observation Y_{t-1}, computes the hidden state $h_t = \{\xi_{0:t}\}^M_{m=1}$, and output $o_t = \{G_\eta(z^m_t)\}^M_{m=1}$. From o_t and the next observation Y_t, the resampling weights w_t are computed. The next hidden state h_{t+1} is a function of this w_t, h_t and the next observation Y_t.

Algorithm 2 Training algorithm

Input : Observations $Y_{0:T}$, Number of particles M, parameter estimate θ.
Initialize $\{(\xi^i_0, \omega^i_0)\}_{1\leqslant i\leqslant M}$.
for $t = 0$ **to** $T-1$ **do**
  for $m = 1$ **to** N **do**
    {*Selection*}
    Sample a trajectory $\xi^{'}_{0:t}$ in $(\xi^m_{0:t})_{1\leqslant m\leqslant N}$ with probability $\{\omega^j_t\}_{1\leqslant j\leqslant N}$.
    {*Mutation - Create a new particle and append it to the resampled ancestral trajectory*}
    Sample a particle ξ^m_{t+1} through a forward pass of the SMC transformer with the ancestral trajectory $\xi^{'}_{0:t}$.
    Update the ancestral line, $\xi^m_{0:t+1} \leftarrow (\xi^{'}_{0:t}, \xi^m_{t+1})$.
    Set $\omega^m_{t+1} \leftarrow p_\theta(Y_{t+1}|\xi^m_{t+1})$.
  end for
  Set $\omega_{t+1} \leftarrow \mathrm{Softmax}\{(\omega^m_{t+1})_{1\leqslant m\leqslant M}\}$.
end for
Estimate the log-likelihood gradient $S^M_{\theta,T}$ using (6.6).
Update $(\eta, W^q, W^\kappa, W^v)$ with SGD using $S^M_{\theta,T}$.
Update $(\Sigma^X, \Sigma^q, \Sigma^\kappa, \Sigma^v)$ with EM based on (6.7).

by the following particle-based estimator :

$$Q^M_{\theta,T} = \sum_{m=1}^{M} \omega^m_T \sum_{t=0}^{T} \left[\log p_\theta(\xi^m_t|\xi^m_{0:t-1}, Y_{t-1}) + \log p_\theta(Y_t|\xi^m_t)\right] \ .$$

Then, $Q^M_{\theta,T}$ may be maximized with respect to all covariances to obtain the new estimates. This is a straightforward update which yields for instance for Σ^X for the p-th update :

$$\Sigma^{Y,p} = \frac{1}{T+1} \sum_{m=1}^{M} \omega^m_T \sum_{t=0}^{T} (Y_t - G_\eta(z^m_t))^\top (Y_t - G_\eta(z^m_t)) \ , \tag{6.7}$$

where z^m_t are the resampled particles at time t. The new estimate of Σ^X is then $\widehat{\Sigma}^Y = (1-\eta_p)\widehat{\Sigma}^Y + \eta_p \Sigma^{Y,p}$ where γ_p is a learning rate chosen by the user ($\gamma_p = p^{-0.6}$ in the experiments).

Algorithm 3 Inference algorithm

Input : Observations $Y_{0:t}$, parameter estimate $\widehat{\theta}$, number of samples N for the predictive distribution.
Output : an empirical predictive distribution $\widehat{Y}_{t+1} = \{\hat{Y}^1_{t+1}, ..., \hat{Y}^N_{t+1}\}$.
Sample M weighted particles $\{(\xi^m_{0:t}, \omega^m_t)\}_{1 \leqslant m \leqslant N}$ through a forward pass of the SMC Transformer on $Y_{0:t}$
for $n = 1$ **to** N **do**
  Sample a trajectory $\xi^{'}_{0:t}$ in $(\xi^m_{0:t})_{1 \leqslant m \leqslant N}$ with probability $\{\omega^j_t\}_{1 \leqslant j \leqslant N}$.
  Sample $\widehat{z}^n_{t+1}$, with distribution $p_{\widehat{\theta}}(z_{t+1}|\xi^{'}_{0:t}, Y_t)$
  Sample $\widehat{Y}^n_{t+1}$ from the Gaussian distribution with mean $G_\eta(\widehat{z}^n_{t+1})$ and variance Σ^Y.
end for

Particle Filtering In Algorithm 2, for all $t \geqslant 1$, once the observation Y_t is available, the current weighted particle sample $\{(\omega^m_t, \xi^m_{0:t})\}^N_{m=1}$ is transformed into a new weighted particle sample. This update step is carried through in two steps, *selection* and *mutation*, as explained for instance in [224]. This procedure introduced in [145, 58] approximates the joint smoothing distributions of the latent states given the observations, by using the genealogy of the particles produced by the auxiliary particle filter. The genealogical trajectories are defined recursively and updated at each time step with the particle ξ^m_{t+1}. As a result, at each time step, the algorithm selects an ancestral trajectory with the weights $\{\omega^m_t\}^N_{m=1}$ and then extends this trajectory using the newly sampled particle ξ^m_{t+1}.

This algorithm maintains a set of weighted particles and associated genealogical trajectories as an estimation of the stochastic latent states, which allows to solve two usual objectives in state-space models : (i) the *state estimation problem*, which aims to recover the latent attention parameter z_t at time t given the observations $Y_{0:T}$, and (ii) the *inference problem* which aims at approximating the distribution of Y_{t+1} given $Y_{0:t}$. The next section focuses on the latter, which provides a natural measure of uncertainty for the SMC Transformer predictions.

6.3.3 Inference and predictive distribution

Given the parameters $\widehat{\theta}$ after training, we solve the inference problem by observing that

$$p_{\widehat{\theta}}(Y_{t+1}|Y_{0:t}) = \int p_{\widehat{\theta}}(Y_{t+1}, z_{0:t+1}|Y_{0:t}) \mathrm{d}z_{0:t+1}$$

is approximated using the weighted samples at time t by

$$\widehat{p}^M_{\widehat{\theta}}(Y_{t+1}|Y_{0:t}) = \sum_{m=1}^{M} \omega^m_t \int p_{\widehat{\theta}}(Y_{t+1}|z_{t+1}) p_{\widehat{\theta}}(z_{t+1}|\xi^m_{0:t}, Y_t) \mathrm{d}z_{t+1} \ ,$$

which can be approximated with Monte Carlo samples as described in Algorithm 3. With such an empirical predictive distribution, it is possible to compute any statistical metric, e.g. mean, confidence interval, percentile. Note that this Monte Carlo estimate can be extended to predictions at future time steps by using the predictions as the next input observations of the inference algorithm.

This inference procedure is computationally efficient as it only requires M particles (or forward passes) to gene-

rate N samples with $M \ll N$, while concurrent methods require one forward pass per sample [94]. It also offers a flexible framework to estimate the predictive distribution. Indeed, the algorithm can be extended to more sophisticated estimation methods than a simple Monte Carlo estimate, that could improve both the predictive performance of the SMC Transformer, and its uncertainty estimation : we leave this for future works.

6.4 Experiments

6.4.1 Experimental Settings and Implementation details

To evaluate the performances of the SMC Transformer, we designed two experimental protocols. First, we create three synthetic datasets with known observation models : the goal is to assess whether the SMC Transformer can capture the true distribution of the observations. Second, we evaluate our model on several real-world datasets on time-series forecasting problems while measuring classic predictive intervals metrics.

For each experiment, we compare the SMC Transformer with the following baselines : a deterministic LSTM [125] and transformer, a LSTM and transformer with *MC Dropout* [94], a Bayesian LSTM [88]. [1]

We implement two versions of the SMC Transformer, a one-layer / one-head network, and for the real-world setting, a 2-layers / 4-heads network. The projection G_η is a point-wise feed-forward network with layer normalization [9] and residual connections as in [9], and as detailed in Section 3.2.2. To ensure full differentiability of the SMC Transformer, we apply the *reparametrization trick* [144] on the Gaussian noises of the self-attention random variables.

The LSTM models (deterministic LSTM, MC Dropout LSTM, Bayesian LSTM) have a number of units in the recurrent layer equal to 32. The Transformer models (deterministic Transformer, MC Dropout Transformer, and SMC Transformer) have a depth (dimension of the attention parameters) equal to 32, and a number of units in the feed-forward neural network that transforms the attention vector also equal to 32. For training the SMC Transformer and the baselines, we use the ADAM algorithm with a learning rate of 0.001 for the LSTM networks and the original custom schedule found in [Vaswani et al., 2017] for the Transformer networks. Models were trained for 50 epochs, except the Bayesian LSTM that was trained for 150 epochs (except for the weather dataset, for which it was trained for 50 epochs). For the two synthetic models, a batch size of 32 was used. On the real-world setting, batch sizes of 32, 64, 256, 128 and 64 were respectively used for the covid, air quality, weather, energy and stock datasets.

Additional details about datasets and baseline models are provided in B.2.

1. We use the blitz github library https://github.com/piEsposito/blitz-bayesian-deep-learning

6.4.2 Estimating the true variability of the observations

Synthetic Data We design three synthetic time-series with a sequence length of 24 observations. For setting I, one data sample $Y = (Y_0, Y_1, ..., Y_{24})$ is drawn as follows, $Y_0 \sim \mathcal{N}(0,\, 1)$ and for $t \geqslant 0$:

$$Y_{t+1} = \alpha Y_t + \sigma \varepsilon_{t+1} \ ,$$

where $(\varepsilon_t)_{1 \leqslant t \leqslant 24}$ are i.i.d standard Gaussian variables independent of Y_0.

For setting II, the law of a new observation given the past is multimodal and drawn as follows, $Y_0 \sim \mathcal{N}(0,\, 1)$ and for $t \geqslant 0$:

$$Y_{t+1} = \alpha U_{t+1} Y_t + \beta(1 - U_{t+1}) Y_t + \sigma \varepsilon_{t+1} \ ,$$

where $(\varepsilon_t)_{1 \leqslant t \leqslant 24}$ are i.i.d standard Gaussian variables independent of Y_0 and $(U_t)_{1 \leqslant t \leqslant 24}$ are i.i.d Bernoulli random variables with parameter p independent of Y_0 and of variance $(\varepsilon_t)_{1 \leqslant t \leqslant 24}$.

Setting III is an ARMA(p,q) model, with $p = 8$ and $q = 4$ giving the following law of observations given the past :

$$Y_{t+1} = \sum_{i=0}^{p-1} \alpha_i Y_{t-i} + \varepsilon_{t+1} + \sum_{j=0}^{q-1} \sigma_j \varepsilon_{t-j} \ ,$$

with $(\varepsilon_{t-i})_{0 \leqslant i \leqslant q}$ i.i.d standard Gaussian variables.

In setting I, the dataset is sampled with $\alpha = 0.8$ and $\sigma^2 = 0.5$. In setting II, the dataset is sampled with $\alpha = 0.9$, $\beta = 0.6\alpha$, $p = 0.7$ and $\sigma^2 = 0.3$. In setting III, the auto-regressive and moving-average parameters of the ARMA model are respectively $(\alpha_i)_{0 \leq i \leq p-1} = (0.75, -0.25, 0.095, -0.07, 0.05, -0.015, 0.01, 0.0075)$ and $(\sigma_i)_{0 \leq i \leq q-1} = (0.65, 0.35, -0.1, 0.08)$.

To evaluate if the SMC Transformer and the baselines can also correctly capture an observation model with no noise, we also consider settings I et II with a very small σ, i.e $\sigma^2 = 10^{-5}$. The associated results are detailed in Table B.2 of the Appendix.

Metrics We used two metrics to estimate the true variability of the observation for the two synthetic models. First, we compute the Mean Square Error (*mse*) between the true observations and the predicted mean of the observations to measure the model predictive performance. For the SMC Transformer, this corresponds to the mean square error between the weighted mean over the predictions and the ground truth. Secondly, we refer as *dist-mse*, the empirical estimate of the mean square error of the predictive distribution of Y_{t+1} given the past observations for all time steps t. Such an estimate is obtained by generating 1000 samples from the predictive distribution. For the SMC Transformer, they are drawn from the SMC estimate of the law of Y_{t+1} given $Y_{0:t}$ as detailed in Algorithm 3. For the baselines, they are drawn by performing 1000 stochastic forward passes on each data sample. For setting

TABLE 6.1 – Mean Square Error of the mean predictions (mse) and Mean Square Error of the predictive distribution (dist-mse) on the test set versus the ground truth, for settings I, II and III. Values are computed with a 5-fold cross-validation procedure on each dataset. Standard deviations are displayed in parenthesis when they are larger than 0.01. For the LSTM and Transformer models with MC Dropout, p is the dropout rate. For the Bayesian LSTM, M is the number of Monte Carlo samples to estimate the ELBO loss [88]. For the SMC Transformer, M is the number of particles of the SMC algorithm.

	Setting I		Setting II		Setting III	
	mse	dist-mse	mse	dist-mse	mse	dist-mse
True Model	0.5	0.50 (0.03)	0.3	0.35 (0.07)	-	1.56
LSTM	**0.50**	N.A	0.32	N.A	0.20 (0.02)	N.A
Transformer	0.52	N.A	0.32	N.A	0.26 (0.02)	N.A
LSTM drop.						
$p = 0.1$	0.48	0.004	0.32	0.003	0.19 (0.05)	1.09 (0.05)
$p = 0.5$	0.53	0.03	0.33	0.02	0.22 (0.02)	1.23 (0.02)
Transf. drop.						
$p = 0.1$	**0.50**	0.02	0.31	0.03	0.26 (0.03)	1.03 (0.04)
$p = 0.5$	0.52 (0.01)	0.05 (0.02)	0.33 (0.02)	0.05 (0.02)	0.38 (0.03)	1.07 (0.03)
Bayes. LSTM						
$M = 10$	0.53 (0.01)	0.03	0.37 (0.01)	0.04	0.35 (0.01)	1.11 (0.03)
SMC Transf.						
$M = 10$	0.52	**0.49**	**0.30**	**0.35**	0.35	**1.35** (0.04)
$M = 30$	0.49	0.52	0.34	**0.35**	0.35	1.32 (0.02)

I, the *dist-mse* measure is given by $\mathbb{E}[(Y_{t+1} - \alpha Y_t)^2 | Y_t]$, and the true value is $\sigma^2 = 0.5$. For setting II, the measure is : $p\mathbb{E}[(Y_{t+1} - \alpha Y_t)^2 | Y_t] + (1-p)\mathbb{E}[(Y_{t+1} - \beta Y_t)^2 | Y_t]$, for which the true value is 0.35. For setting III, the measure is $\mathbb{E}[(Y_{t+1} - \sum_{i=0}^{p-1} \alpha_i Y_{t-i} | Y_t, Y_{t-1}, ..., Y_{t-p}]$, for which the true value is $1 + \sum_{i=0}^{q-1} \alpha_i^2 = 1.56$.

Results Experimental results are summarized in Table 6.1 and Figure 6.2. In Table 6.1, we observe that all models perform similarly when predicting the mean of the observations. Yet, only the SMC Transformer manages to capture the true distribution of the observations accurately, with a *dist-mse* measure close to the ground truth. On the other side, both LSTM with MC Dropout and the Bayesian LSTM highly underestimate the variability of this distribution for settings I and II, as illustrated by the small values of their *dist-mse*. For setting III, such models give a better estimate of the variability of the observation model, but still remain below the one estimated by the SMC Transformer. Such findings are also illustrated in Figure 6.2. We there display the empirical predictive distribution of the different methods based on the 1000 samples versus the true 95% confidence interval given by the known observation model for the 24 timesteps of a test sample from setting I. Again, the SMC Transformer tends to match the true variability of the observations while concurrent methods clearly underestimate it.

6.4.3 Predictive Intervals on real-world time-series.

We evaluate the performance of the stochastic Transformer on five real-world sequence prediction problems using the Covid-19 [2], Jena weather [3], the GE stock [4] datasets, and the air quality and energy consumption data

2. https ://github.com/CSSEGISandData/COVID-19
3. https ://www.bgc-jena.mpg.de/wetter/
4. https ://www.kaggle.com/szrlee/stock-time-series-20050101-to-20171231

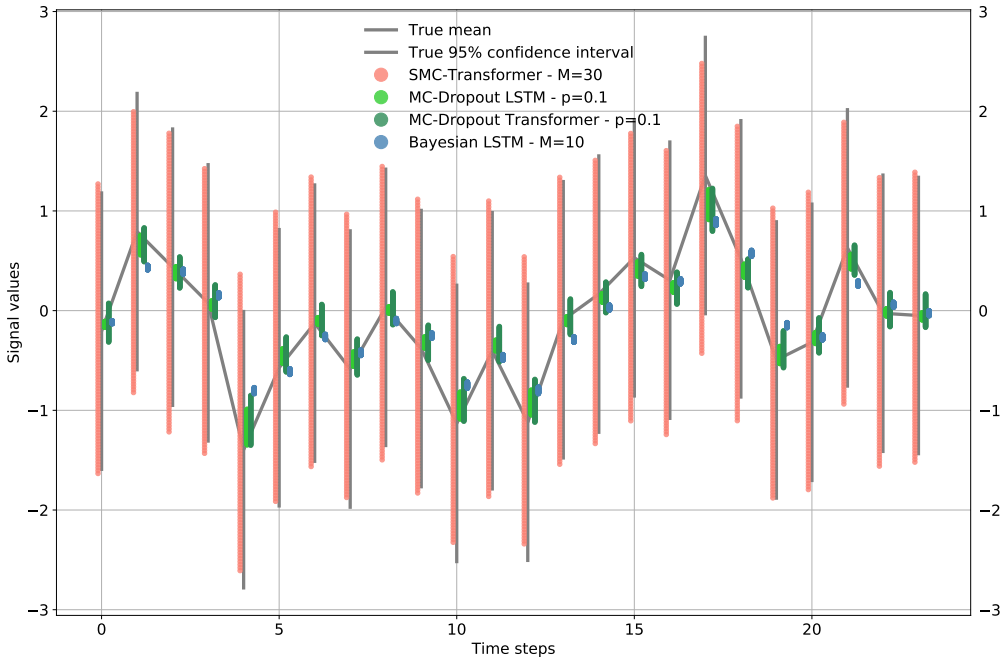


FIGURE 6.2 – Samples distribution on a test example (Setting I).

from the UCI repository [5]. Each dataset is split between a train dataset containing 70% of the data samples, and a validation and test sets containing an equal number of the remaining 15% of the data samples. Further details are available in B.2.

For each of these time-series, we both perform *unistep forecast* and *multistep forecast*. Unistep forecast estimates the conditional distribution $p_{\hat{\theta}}(Y_{t+1}|Y_{0:t})$ for every timestep of every test sample. The multistep forecast estimates the predictive distribution $p_{\hat{\theta}}(Y_{\tau_h+t}|Y_{0:\tau_h})$, with $1 \leqslant t \leqslant \tau_F$ on each test sample, given a frozen history of τ_h timesteps and a number τ_F future timesteps to predict.

Predictive intervals metrics In real-world time-series, we do not have access to the true distribution of the observations : we thus assess uncertainty by computing the Predicted Interval Coverage Percentage (PICP) [220]. For any time step t of the test set, a generative model can provide a lower and upper predicted interval (PI) bound, respectively $\hat{x}_{L_t}$ and $\hat{x}_{U_t}$ by sampling N predictions of the predictive distribution at time t. The PICP [220] of the true observations is then defined as :

$$\text{PICP} = \frac{1}{n}\sum_{i=1}^{n} k_i \quad \text{with} \quad k_i = \begin{cases} 1 & \text{if } \hat{x}_{L_i} \leqslant Y_i \leqslant \hat{x}_{U_i}\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

5. https ://archive.ics.uci.edu/ml/datasets

TABLE 6.2 – One-step mse, and multistep forecast PICP and MPIW. The underlined values correspond to the best performances. MPIW are only effective when the PICP is valid (above 0.95 - green). MPIW associated with poor PICP (below 0.90 - red) are grayed. For the transformer architectures, L is the number of layers and H is the number of heads.

	Covid		Air quality		Weather		Energy		Stock	
	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw
LSTM drop.										
$p = 0.1$	0.150	0.67 | 0.61	0.139	0.54 | 0.73	0.097	0.61 | 1.11	0.070	0.96 | 1.12	0.065	0.85 | 0.74
$p = 0.5$	0.155	0.80 | 1.64	0.211	0.70 | 1.31	0.165	0.75 | 1.68	0.218	0.88 | 1.57	0.112	0.87 | 1.42
Transf. drop.										
$p = 0.1$, $L = 1$, $H = 1$	0.121	0.74 | 0.74	0.141	0.69 | 1.44	0.130	0.42 | 0.67	0.046	0.89 | 0.60	0.076	0.71 | 0.54
$p = 0.5$, $L = 1$, $H = 1$	0.208	0.84 | 1.52	0.196	0.77 | 1.97	0.209	0.61 | 1.11	0.090	0.94 | 1.21	0.106	0.83 | 0.77
Bayesian LSTM	0.144	0.15 | 0.23	0.192	0.49 | 0.72	0.113	0.36 | 0.54	0.121	0.93 | 1.08	0.086	0.34 | 0.45
SMC Transf.										
$M = 10$, $L = 1$, $H = 1$	0.128	0.91 | 1.85	0.148	0.97 | 3.17	0.181	0.92 | 2.93	0.043	0.97 | 1.33	0.071	0.98 | 1.80
$M = 10$, $L = 2$, $H = 4$	0.153	0.89 | 1.20	0.122	0.90 | 2.54	0.126	0.93 | 2.82	0.041	0.98 | 1.30	0.063	0.96 | 1.34

where the mean is computed over all time steps considered in the test set. The Mean Predicted Interval Width (MPIW) is :

$$\text{MPIW} = \frac{1}{n}\sum_{i=1}^{n}(\hat{x}_{U_i} - \hat{x}_{L_i}) \ .$$

If $\hat{x}_L$ and $\hat{x}_U$ represent the predictive bounds of a $(1-\alpha)$ confidence interval, intuitively, we want the associated PICP_α to capture $1-\alpha$ proportion of the true observations.

Results Table 6.2 presents the tuple $(\text{PICP}_{0.05}, \text{MPIW}_{0.05})$ associated with a 95% confidence interval when performing multistep forecasting on the five datasets. Similarly to Section 6.4.1, we also report the *mse* when performing unistep forecasting over the test set between the mean predictions and the true observations. For $(\text{PICP}_{0.05}, \text{MPIW}_{0.05})$, the highest $\text{PICP}_{0.05}$ gives the best measure when it is lower than 0.95 ; otherwise, the lowest $\text{MPIW}_{0.05}$ gives the best measure, as proposed in [220].

Again, all approaches present similar performances in terms of *mse* values : while the training algorithm of the SMC Transformer does not optimize such metric but the log-likelihood of the observations, the model still has a predictive performance competitive with state-of-the-art sequential neural networks. When looking at uncertainty metrics, the SMC Transformer outperforms such baselines in terms of $(\text{PICP}_{0.05}, \text{MPIW}_{0.05})$ for all datasets, except the energy consumption data, for which it is slightly outperformed by the MC Dropout LSTM with dropout rate equal to 0.1.

Figure 6.3 represents the evolution of the $\text{PICP}_{0.05}(t)$ per timestep t when doing multistep forecasting for four of the five datasets : the SMC Transformer gives higher $\text{PICP}_{0.05}(t)$ and tends to have a more stable PICP evolution over time than the concurrent baselines. Moreover, among the other approaches, there is no clear second best model for predicting uncertainty : sometimes the SMC Transformer is trailed by the MC Dropout Transformer, sometimes by the MC Dropout LSTM. The Bayesian LSTM tends to be particularly overconfident with PICP values often much lower than the ideal 95% threshold. As for the MC Dropout models, higher uncertainty measures (obtained

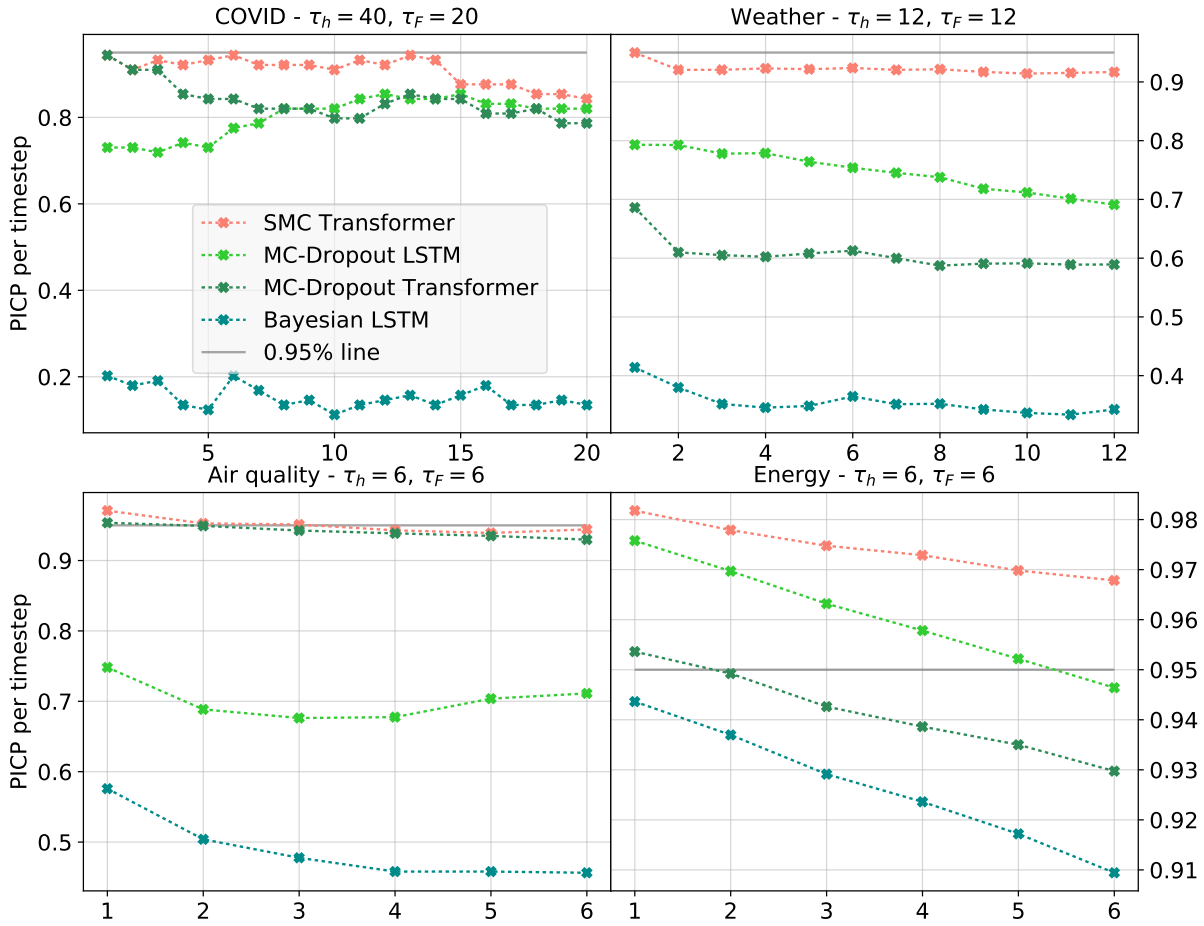


FIGURE 6.3 – Plot of PICP per timestep when doing multi-step forecast for four of the five datasets. τ_h represents the number of past timesteps used to predict each of the τ_F future timesteps.

generally with higher dropout rates) comes at the expense of predictive performance degradation.

The discrepancy in uncertainty measures between the SMC Transformer and the baselines tends to be higher for datasets with longer sequences, suggesting that our approach is well-suited to model complex structured predictions problems with long-range dependencies. For instance, the stock dataset with a long temporal dependency of 40 past timesteps gives a gap of 11% between the SMC Transformer and the second best model. However, this gap is only equal to 1% for the energy dataset, which depends only on 12 past timesteps. The deep SMC Transformer version ($L = 2$, $H = 4$) displays a better predictive performance for multivariate time-series (all datasets except the covid), and tend to be less overconfident than the shallow version. Additional results with deterministic baselines, complementary MC Dropout LSTM and Transformer architectures, and the unistep forecast case are available in B.2.

6.4.4 Particles degeneracy over time

As highlighted in [145, 146, 82, 226], the particle smoother based on the genealogical trajectories suffers from the path degeneracy issue. At each time $t \geqslant 1$, the first step to build a new trajectory is to select an ancestral trajectory chosen among M existing trajectories : as the number of resampling steps increases, the number of ancestral trajectories which are likely to be discarded increases.

In Figure 6.4, we illustrate this degeneracy phenomena on the covid dataset for a SMC Transformer with 60 particles. The figure displays the number of unique resampled trajectories $\xi_{0:t-1}^m$ remaining for the 60 timesteps of the sequential process, averaged over the test set. The further we go in the past, the more the trajectories degenerate : for the first five timesteps, the trajectories are derived from only 2 unique particles, and only the last ten timesteps present a set of unique particles whom size is superior to one sixth of the original size (60).

Yet, as illustrated in previous section, such degeneracy does not impact severely the empirical performances of the SMC Transformer. Moreover, there exist solutions to improve the approximation $S_{\theta,T}^M$ and to avoid such phenomena. A simple approach consists in using a fixed-lag smoother of [210] : for each $0 \leqslant t \leqslant n$, the trajectories $\xi_{0:t-1}^m$ are only resampled up to a few time steps δ after t.

Figure 6.4 indicates which value of δ should be used if we want to keep a sufficiently large number of unique past trajectories, e.g. $\delta = 10$. Other approaches based on the decomposition of the smoothing distributions using backward kernels have been widely studied in the hidden Markov models literature [70, 105, 59, 212]. Extending such approaches to deep learning architectures at a reasonable computational cost remains a practical challenge.

6.5 Related work

Uncertainty estimation in Deep Learning has sparked a lot of interest from the research community over the last decade, leading to a rich literature on the subject. Such works are usually divided between frequentist ap-

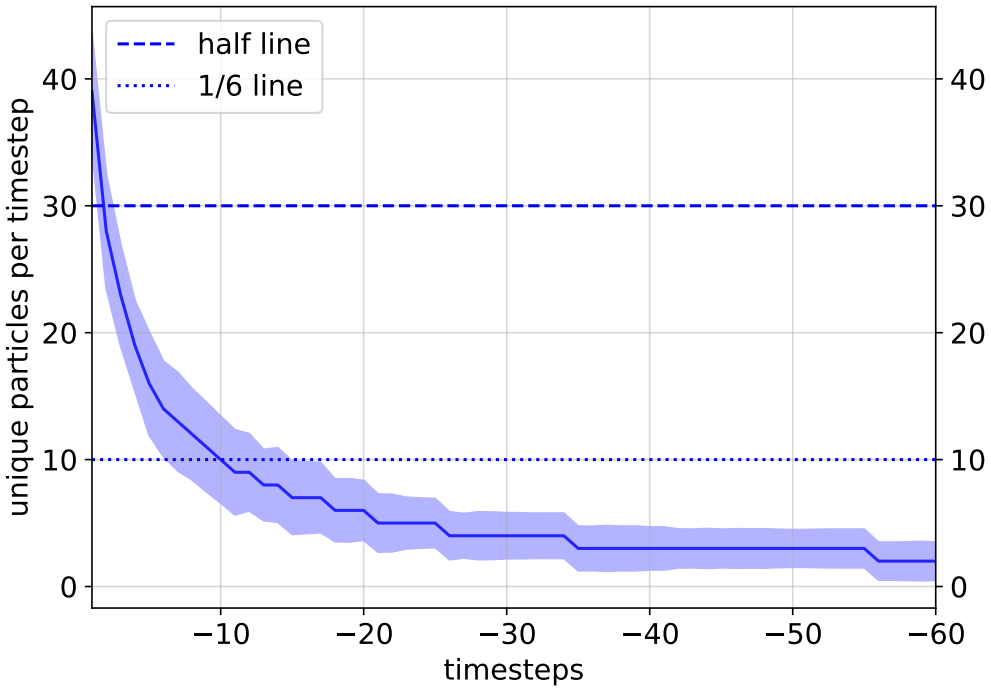


FIGURE 6.4 – Evolution of the number of unique particles (or non-degenerated particles) over the past timesteps for the resampled trajectories $\xi^m_{0:t-1}$ for a SMC Transformer with 60 particles, on the covid dataset. The timesteps are labeled from the most recent one $t-1$ (labeled as -1 on the figure), to the first timestep (labeled as -60 on the figure). The straight line is the average number of particles over the predicted trajectories on the test set, while the shadow area is the corresponding 95% confidence interval.

proaches [152, 128, 220, 271, 280, 213] and Bayesian ones. Among the latter, Bayesian Neural Networks (BNNs) as defined in [93] put a prior distribution over the networks weights [27, 94, 139, 4, 274, 123]. However, they suffer from several limitations, one being their inability to correctly assess the posterior distribution [87] as illustrated in Section 6.4, the other being their computational overhead. Although MC Dropout is one of the most scalable Bayesian inference algorithms, its sampling procedure at inference, that relies on one stochastic forward pass per sample from the predictive distribution, is more computationally expensive than the SMC Transformer, for which sampling comes from a Gaussian mixture model directly derived from the particles predictions.

RNNs with stochastic latent states The SMC Transformer is part of an emerging line of research bridging state space models (historically restricted to simpler statistical models such as Hidden Markov Models or Linear Gaussian Models) and Deep Neural Networks. A few works have proposed recurrent neural networks with stochastic latent states, such as VRAE [76], VRNN [49], SRNN [90], or VHRNN [63]. Additionally, several stochastic attention models [64, 255, 14] have been developed in sequence-to-sequence architectures to improve natural language generation. Yet, they are all based on soft or hard attention mechanisms, and does not scale easily to more complex attention models such as a Transformer network. Such models use training algorithms that rely on variational inference methods [25] to approximate the intractable posterior distribution over the latent states. Such learning procedures are popular and are computationally efficient, but output a predictive distribution known to be ill-fitted for estimating

certain families of distributions, such as for instance multimodal distributions.

SMC methods and RNNs Several SMC algorithms have then been developed to get a better estimator of the marginal likelihood of the observations for such stochastic RNNs, again in a variational inference framework. [157, 203, 191, 189] proposed particle filtering algorithms, while [153, 198] developed particle smoothing ones. Our work differs from the models and algorithms mentioned above in several ways. First, the training algorithm of the SMC Transformer relies on the Fisher's Identity to estimate the gradient of the likelihood of the observations, instead of a variational objective. Secondly, this is the only work proposing : (i) a novel recurrent generative model based on a stochastic self-attention model, and (ii) a novel SMC algorithm to estimate the posterior distribution of the unobserved states, with resampling weights directly depending on the output of the SMC Transformer. Finally, while the above works only leverage the SMC algorithm to get a better and lower-variance estimator of the marginal log-likelihood, our work and evaluation protocol focus on uncertainty measurements for sequence prediction problems.

Time-series uncertainty quantification Neural models may also be extended to quantify uncertainty in multi-step time-series forecasting by optimising a quantile loss. This approach has been applied on both recurrent networks [244, 285] and Transformers [171, 174]. Although easier to train, these methods do not output a full predictive distribution, making them restricted to specific uncertainty metrics and regression problems. In this chapter, we train a generative model, which gives a generic framework to reason about uncertainty. Therefore, it can be seamlessly adapted to various other uncertainty use-cases (active learning [99], etc), and types of sequential data (natural language processing [292], etc.).

6.6 Conclusion

In this chapter, we proposed a novel recurrent network that naturally captures the distribution of the observations. This model maintains a distribution of self-attention parameters as latent states, estimated by a set of particles. It thus outputs a distribution of predictions instead of a single-point estimate. Our inference method gives a flexible framework to quantify the variability of the observations. To our knowledge, this is the first method proposing a generative model for the transformer network, and one of the few focusing on uncertainty quantification in the context of sequence prediction. Moreover, this SMC Transformer layer could be used as a "plug-and-play" layer for uncertainty quantification in a deeper neural network encoding sequential data.

The limitations of our model are (i) its computational overhead at training time, and (ii) the fact that it relies on the most basic smoothing algorithm, the poor man smoother, which in particular leads to the particle degeneracy phenomena observed in Section 6.4.4. Answering to this two limitations could provide a promising extension to (i) scale our current approach on large-scale text datasets, and (ii) better take in account the long-range dependencies in NLP data with sophisticated smoothing techniques. In that perspective, in Chapter 7, we develop on online

smoothing algorithm with lesser computational complexity than other comparable algorithms, that further attempts to scale SMC Methods for Deep Generative Modelling.

6.7 Application of the SMC Transformer to Language Modelling

In this section, we explore the application of the SMC Transformer in a NLP setting on Language Modelling tasks. We first describe the model extension for such setting in section 6.7.1, then describe the task and evaluation metrics considered in section 6.7.2, and finally display the experimental results in section 6.7.3.

6.7.1 Extension of the Model for Language Modelling

The SMC Transformer learns a Language Model $p_\theta(w_{t+1}|w_{0:t})$ for $0 \leq t \leq T$ given stochastic latent attention parameters $X_{0:T} = (z_{0:T}, q_{0:T}, \kappa_{0:T}, v_{0:T})$.

The stochastic self-attention model. We use the same self-stochastic attention model described in Equation 6.4, i.e we add a centered Gaussian noise to the parameters (q, κ, v, z) of the transformer self-attention model, with variance respectively referred as $(\Sigma^q, \Sigma^\kappa, \Sigma^v, \Sigma^z)$. We consider multi-dimensional Gaussian noises with diagonal covariance matrices : $(\Sigma^q, \Sigma^\kappa, \Sigma^v, \Sigma^z)$ are in $\mathbb{R}^{d\times d}$, with d the dimension of the attention parameters. Their diagonal elements are parametrized by their logarithm and learned through stochastic gradient descent with the other parameters of the network.

The observation model. In a classification setting, the observation model is simply the multinomial distribution over a vocabulary $\mathcal{V}$ produced by a softmax transformation of the SMC Transformer output $G_\mu(z_t)$:

$$w_t \sim \text{softmax}(G_\mu(z_t))$$

Training procedure. Using a set of weighted particles (ξ_t^m, ω_t^m) to approximate the latent states $X_t = (z_t, q_t, \kappa_t, v_t)$, the score function is the same as described in section 6.3.2 :

$$S_{\theta,T}^M = \sum_{m=1}^{M} \omega_T^m \sum_{t=0}^{T} \left[\nabla_\theta \log p_\theta(\xi_t^m|\xi_{0:t-1}^m, w_{t-1}) + \nabla_\theta \log p_\theta(w_t|\xi_{0:t}^m, w_{t-1})\right] , \tag{6.8}$$

In a classification setting, $\log p_\theta(w_t|\xi_{0:t}^m, w_{t-1})$ is the categorical cross-entropy. At time step t of the particle filtering algorithm, the unnormalized filtering weights $(\tilde{\omega}_t^m)_{m=1}^M$ are computed using the observation model :

$$\tilde{\omega}_t^m = \text{softmax}(G_\mu(z_t^m))[w_t]$$

where w_t is the ground-truth next token. Then the normalized filtering weights are computed as a softmax over the unnormalized weights.

Inference procedure. At inference, from an input prompt $\text{prompt} = w_{<t}$ containing a truncated words sequence from the test set, we are interested in generating M possible text continuations (with M corresponding to the number of particles), as follows :

1. At the first timestep $t+1$ of the decoding process, select M next word probability distribution $\{p_{t+1}^j\}_{j=1}^M$ from $(p_{t+1}^m)_{m=1}^M$ using the last resampling weights $(\omega_t^m)_{m=1}^M$:

$$I_{t+1}^j \sim \text{Multinomial}(\omega_t^m)_{m=1}^M$$
$$p_{t+1}^j = p_{t+1}^{I_{t+1}^j}$$

   Then sample one word per selected probability distribution.

2. For the following timesteps $t' > t+1$ of the decoding process, we follow the same process except that the M probability distributions are sampled from a multinomial distribution with weights $(1/M, ..., 1/M)$. Indeed, at inference, we do not have access to the ground-truth tokens, and hence there is no obvious resampling process for the particles.

Resampling trajectories at inference using an external language metric. Intuitively, generating text at inference with the SMC Transformer might lead to degenerated text, as the sequence of words being produced grows longer. Indeed, there is no resampling process to discard particles that could lead to inconsistent text : such particles are propagated, leading potentially to mistakes that can accumulate quickly. To improve text generation, we thus propose to introduce a resampling mechanism for the particles produced by the trained SMC Transformer at inference based on an external language metric. We chose the perplexity of GPT-2 on the generated text sequence for such a metric. At timestep t of the decoding process, given the M sequence of past words $\{w_{0:t-1}^m\}_{m=1}^M$ generated by a SMC Transformer, the score function computing the M resampling weights $\{\omega_t^{m,\text{inf}}\}_{m=1}^M$ is based on the perplexity of GPT-2 on the top-10 words produced by each probability distribution $p(.|w_{0:t-1}^m)$:

$$\begin{aligned}
&\text{Get the top-10 words and their probability from } p(.|w_{0:t-1}^m) : \quad [w_t^{m,j}, \alpha_t^{m,j}] = \text{top}_{10}(p(.|w_{0:t-1}^m)); \\
&\text{Compute GPT-2 perplexity on } (w_{0:t-1}, w_t^{m,j}) : \quad \text{gpt2ppl}(w_{0:t-1}, w_t^{m,j}); \\
&\text{Compute unnormalized resampling weights : } \quad \{\tilde{\omega}_t^{m,\text{inf}}\}_{m=1}^M = \Big\{ \sum_{j=1}^{10} \alpha_t^{m,j} * \text{gpt2ppl}(w_{0:t-1}, w_t^{m,j}) \Big\}_{m=1}^M; \\
&\text{Normalize resampling weights : } \quad \{\omega_t^{m,\text{inf}}\}_{m=1}^M = \text{softmax}(\{\tilde{\omega}_t^{m,\text{inf}}\}_{m=1}^M).
\end{aligned} \tag{6.9}$$

6.7.2 Experimental Setting

Dataset. We evaluate the SMC Transformer on a Language Modelling task using the ROC story dataset[6] made of short 5-sentences stories. The original dataset contains 52,665 stories of 5 sentences. We extracted the first 2 sentences of the 5-sentences stories, as a shorter dataset for Language Modelling. Then, we filtered text samples with maximum length equal to 20 words. The final dataset split contains 24,024 text samples for the train dataset, and 5000 text samples for both the validation and test dataset. The vocabulary contains 20,110 words.

Models. We compare our model to a classic Transformer [277] with various rates of dropout (see Section 3.2.2 for the dropout layers within the Transformer architecture), from no dropout, through a rate of 0.1, and a rate of 0.5. We kept the model with the best validation loss. For the SMC Transformer, we evaluate multi-dimensional versus one-dimensional diagonal variance settings, i.e the former has different diagonal elements on the covariance matrix, while the latter has a covariance matrix equal to $\sigma * \mathcal{I}_d$, where $\mathcal{I}_d$ is the identity matrix of dimension $d \times d$. We refer to as $\mathrm{SMC} - \mathrm{T}(\sigma = \sigma_0)$ for the one-dimensional setting, and we refer to as $\mathrm{SMC} - \mathrm{T}(\sigma_{\mathrm{multi}} = \sigma_0)$ for the multi-dimensional setting with initial values for the two settings equal to $\sigma_0 * \mathcal{I}_d$. We also evaluated a SMC Transformer variant with a resampling mechanism at inference (as described in Equations 6.9), referred as $\mathrm{SMC} - \mathrm{T}(\sigma = \sigma_0)(\mathrm{resample})$. As a comparison for the SMC Transformer with a resampling mechanism, we also evaluate the text samples from the baseline transformer using a resampling scheme : for each input sentence from the test dataset, the model decodes 50 text samples using sampling with temperature, and then we use also the perplexity of GPT-2 to select the top-10 text samples. In the results table, we refer to this setting as $\mathrm{Transformer}(\mathrm{resample})$.

Model dimensions and hyper-parameters. We trained one-layer Transformers, with model dimensions (dimension of the words embeddings, dimension of the attention parameters and number of units in the feed-forward neural network of the residual layers) of 128 for the baselines and the SMC Transformer. They are trained for 40 epochs, using the adam optimizer [141] with custom schedule and initial learning rate of 10^{-3} from the original Transformer model [277]. The SMC Transformer has 10 particles. At inference, the trained language models generate text using sampling decoding with temperature equal to 0.7.

Evaluation metrics. For evaluating the trained language models, we first measure their cross-entropy and perplexity on the validation dataset, referred to respectively as 'val-ce' and 'val-ppl' in the results table. For the SMC Transformer, the cross-entropy and the perplexity are computing from a weighted average of the M probability distributions $(p_t^m)_{m=1}^M$ given by each particle :

$$p_t = \sum_{m=1}^{M} \omega_t^m p_t^m,$$

6. https://cs.rochester.edu/nlp/rocstories/

where $(\omega_t^m)_{m=1}^M$ are the particle filtering weights at time step t. Secondly, at inference, we generate text continuations from samples from the test dataset. We feed as text prompt the first sentence of the 2-sentences story test sample, and we ask the trained language model to decode the second sentence. From these text continuations, we first measure the average BLEU score [214] between the generated text sequence and the true sentence (from the test dataset). Secondly, as a measure of language diversity, we compute the self-BLEU score between the ten generated text continuations. As an additional measure for language quality, we compute the average perplexity of GPT-2 [232] on the generated text sequences, referred to as 'gpt2-ppl' in Table 6.3.

6.7.3 Results

Table 6.3 displays the results when evaluating the Language Models on the ROC dataset. On such a setting (i.e natural language with large vocabulary of 20k words), the SMC Transformer actually outperforms largely a deterministic transformer, with better validation cross-entropy and validation perplexity, and slightly better bleu score.

This is confirmed by Figure 6.4 which displays the text samples produced by a classic Transformer versus variants of the SMC Transformer with a resampling mechanism at inference. The samples generated by the SMC Transformer variants tend to be more diverse, while also producing better language. Indeed, when looking at the first example that continues the sentence "Tim was out shopping.", the baseline Transformer generates five samples all starting with "He decided to" that all ends up with inconsistent and incorrect text. The baseline Transformer with a resampling mechanism at inference produces samples slightly more diverse, yet still missing semantic coherence. On the other hand, the SMC Transformer generates more diverse beginnings (e.g "He was watching", "Everyone dared", "He wanted to"), with also more correct text, even if the continuations tend to be poorly grounded with the first sentence. The same trend is observed on the samples continuing the sentence "Allie went to get a perm.". In this example, the baseline Transformer versions (with or without resampling when decoding text) tend to repeat the same words and beginnings of sentence, while the SMC Transformer samples are more diverse and more correct, both semantically and syntactically.

As expected, adding a resampling mechanism based on an external language metric is crucial for the SMC Transformer to generate meaningful and correct text at inference, as illustrated in Figure 6.5. In the figure, we indeed observe that the samples from the SMC Transformer with no resampling mechanism at inference lead to degenerated and incorrect text. On the other hand, as soon as we add the resampling mechanism, the trained SMC Transformer is able to generate syntactically correct samples, yet which tend to lack semantic consistency with the input sentence.

models	val-ce	val-ppl	bleu	gpt2-ppl	selfbleu
$\mathrm{Transformer}$	6.73	840	0.07	672	0.61
$\mathrm{Transformer(resample)}$	-	-	0.08	374	0.68
$\mathrm{SMC-T}(\sigma = 0.1)$	**3.76**	**43**	**0.10**	589	0.65
$\mathrm{SMC-T}(\sigma = 0.5)$	3.81	45	**0.10**	565	0.67
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.1)$	3.91	50	0.09	604	0.62
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.5)$	3.77	44	**0.10**	569	0.66
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.1)\mathrm{(resample)}$	-	-	0.08	**279**	**0.59**
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.5)\mathrm{(resample)}$	-	-	**0.10**	303	0.64

TABLE 6.3 – Results on ROC dataset after training for 40 epochs, for a recurrent Transformer (the baseline) and several configurations of the SMC Transformer with 10 particules, for model dimensions equal to 128.
$\mathrm{Transformer}$ and $\mathrm{Transformer(resample)}$ refer respectively to baseline transformers with or without resampling mechanism (using GPT-2 perplexity) when decoding samples at inference. $\mathrm{SMC-T}(\sigma = \sigma_0)$ and $\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = \sigma_0)$ refer respectively to a SMC Transformer with one-dimensional and multi-dimensional diagonal covariance matrices for the Gaussian noises of the stochastic self-attention model (as defined previously), with initial value equal to $\sigma_0 * \mathcal{I}_d$. $\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = \sigma_0)\mathrm{(resample)}$ is a SMC Transformer with multi-dimensional diagonal covariance matrix, which uses at inference the resampling mechanism based on GPT-2 perplexity and described in previous section to select particles.
For the models version with resampling mechanism at inference when decoding text, the values of the validation cross-entropy and perplexity are not provided, as they are the same than the models with the same hyper-parameters and no resampling at inference. Best results are in bold.

6.7.4 Limits of the SMC Transformer applied to Language

Although the SMC Transformer showcases some potential in producing more diversity when generating text at inference than a deterministic Transformer, a lot of challenges remain to be solved for applying it on large-scale text datasets and on complex Natural Language Generation tasks. The first pitfall of the approach is its computational complexity at training, as illustrated in Tables 6.6. The SMC Transformer takes significantly more time to train than a recurrent Transformer, and its computational cost increases significantly as the length of the input text sequence increases. This becomes a real challenge in language modelling settings requiring large networks and numerous text samples to achieve reasonable performances. We would have a lot of difficulties to (i) apply it on large-scale text datasets with long sequences (the ROC stories dataset has been reduced with sentences shorter than 20 words), to (ii) scale it to a larger network (with for example several layers), (iii) to apply it on top of a pretrained language model (for example GPT-2), and (iv) to use it in Reinforcement Learning framework (for instance as the Policy Language Model of chapter 5). Secondly, the learning of the noise parameters within the stochastic self-attention model suffers from sensitivity to parameters initialization. Indeed, when evaluating different initialization $\sigma_0 * \mathcal{I}_d$ for the diagonal covariance matrices $(\Sigma^q, \Sigma^\kappa, \Sigma^v, \Sigma^z)$, the learned matrices do not converge to the same values, as illustrated in Table 6.7. This suggests a lack of identifiability in the learned generative model (different set of parameters lead to the same score function), which means in practice that the parameter σ_0 of the diagonal covariance matrices initialization becomes an hyper-parameter that can be hard to tune.

Such pitfalls suggest to consider a different approach when combining Sequential Monte Carlo Methods within Transformers in a NLP setting. A simpler approach could be to train deterministic Transformers, and add noise in the self-attention model only at inference : in that case, SMC algorithms could be used for the state estimation problem.

INPUT SEQUENCE	**Tim was out shopping**.
GROUND-TRUTH	He was about to pay.
$\mathrm{Transformer}$	Tim was out shopping. He decided to get a new business cream . Tim was out shopping. He decided to get green new with is . Tim was out shopping. He decided to see some new car a to Tim was out shopping. He decided to buy a new school his trash Tim was out shopping. He decided to get a chips school drove games
$\mathrm{Transformer(resample)}$	Tim was out shopping. Yesterday she had been so many friends and had Tim was out shopping. When his friends noticed a swim was very little Tim was out shopping. He paid for the morning his groceries without groceries Tim was out shopping. When his friend 's was old jeans had too Tim was out shopping. He bought a new friend of the house.
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.5)(\mathrm{resample})$	Tim was out shopping. He was watching all night. Tim was out shopping. He was shopping. Tim was out shopping. Everyone dared him . Tim was out shopping. He wanted to the man him to visit him Tim was out shopping. He was nervous.
INPUT SEQUENCE	**Allie went to get a perm**.
GROUND-TRUTH	She was nervous about it though.
$\mathrm{Transformer}$	Allie went to get a perm. She was not hair. Allie went to get a perm. It was going to do it was was Allie went to get a perm. She was going to do it herself. Allie went to get a perm. She was the because. Allie went to get a perm. she was the it herself.
$\mathrm{Transformer(resample)}$	Allie went to get a perm. She was building asked her lady Allie went to get a perm. She was the hairdresser up early to do it. Allie went to get a perm. She was the hairdresser up early to bed. Allie went to get a perm. She was herself. Allie went to get a perm. She was going to the restaurant
$\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = 0.1)(\mathrm{resample})$	Allie went to get a perm. She had a costume. Allie went to get a perm. She had a concert. Allie went to get a perm. She went to the store as she asked for some Allie went to get a perm. She got out. Allie went to get a perm. It wanted something pretty.

TABLE 6.4 – 5 continuations generated for two given input sentences "Tim was out shopping.", and "Allie went to get a perm." for a classic Transformer versus a SMC Transformer.
$\mathrm{Transformer}$ and $\mathrm{Transformer(resample)}$ refer respectively to baseline transformers with or without resampling mechanism (use GPT-2 perplexity) when decoding samples at inference. $\mathrm{SMC-T}(\sigma_{\mathrm{multi}} = \sigma_0)(\mathrm{resample})$ is a SMC Transformer with multi-dimensional diagonal covariance matrix, which uses at inference the resampling mechanism based on GPT-2 perplexity and described in previous section to select particles.

The approach is explored in the next chapter : in this chapter, we do not focus on NLP experiments, but we show that the new smoothing algorithm we introduce can be easily applied for state estimation in Recurrent Neural Networks.

INPUT SEQUENCE	Tim was out shopping.
GROUND-TRUTH	He was about to pay.
$\mathrm{SMC}-\mathrm{T}(\sigma=0.5)$	Tim was out shopping. He accidentally very wanted birthday his Tim was out shopping. He was always over it a Tim was out shopping. He was so he tree. Tim was out shopping. Her was knocked over piling. Tim was out shopping. He talked walking a
$\mathrm{SMC}-\mathrm{T}(\sigma=0.5)(\mathrm{resample})$	Tim was out shopping. He was watching all night. Tim was out shopping. He was shopping. Tim was out shopping. Everyone dared him. Tim was out shopping. He wanted to the man him to visit him Tim was out shopping. He was nervous.

TABLE 6.5 – 5 sentence continuations generated for a given input sequence "Tim was out shopping." for SMC Transformer variants, with and without resampling at inference.

Model	training time per epoch
Recurrent Transformer	323 sec.
$\mathrm{SMC}-\mathrm{T}(\sigma=0.1)$	1085 sec.
$\mathrm{SMC}-\mathrm{T}(\sigma_{\mathrm{multi}}=0.1)$	1120 sec.

TABLE 6.6 – Training time per epoch for a baseline Recurrent Transformer versus several variants of SMC Transformer with 10 particles. The networks have the same size, i.e we consider one-layer Transformers of dimension 128 for the embedding layer, the dimension of the attention parameters, and the number of units in the feedforward network of the residual layers.

Model	converged values for $(\Sigma^q, \Sigma^\kappa, \Sigma^v, \Sigma^z)$			
$\mathrm{SMC}-\mathrm{T}(\sigma=0.05)$	$\Sigma^q=0.05\mathcal{I}_d;$	$\Sigma^\kappa=0.16\mathcal{I}_d;$	$\Sigma^v=0.34X\mathcal{I}_d;$	$\Sigma^z=1.58\mathcal{I}_d$
$\mathrm{SMC}-\mathrm{T}(\sigma=0.1)$	$\Sigma^q=0.1\mathcal{I}_d;$	$\Sigma^\kappa=0.36\mathcal{I}_d;$	$\Sigma^v=0.56\mathcal{I}_d;$	$\Sigma^z=3.74\mathcal{I}_d$
$\mathrm{SMC}-\mathrm{T}(\sigma=0.5)$	$\Sigma^q=0.5\mathcal{I}_d;$	$\Sigma^\kappa=1.43\mathcal{I}_d;$	$\Sigma^v=2.85\mathcal{I}_d;$	$\Sigma^z=17.2\mathcal{I}_d$

TABLE 6.7 – learned values for the covariance matrices $(\Sigma^q, \Sigma^\kappa, \Sigma^v, \Sigma^z)$ for the one-dimensional SMC Transformer variants, when varying the initialization. $\mathrm{SMC}-\mathrm{T}(\sigma=\sigma_0)$ refers to diagonal covariance matrices initialized to σ_0 times the Identity Matrix $\mathcal{I}_d$.

Chapitre 7

Backward importance sampling for online estimation of state space models

This chapter proposes a new Sequential Monte Carlo algorithm to perform online estimation in the context of state space models when either the transition density of the latent state or the conditional likelihood of an observation given a state is intractable. In this setting, obtaining low variance estimators of expectations under the posterior distributions of the unobserved states given the observations is a challenging task. Following recent theoretical results for pseudo-marginal sequential Monte Carlo smoothers, a pseudo-marginal backward importance sampling step is introduced to estimate such expectations. This new step allows to reduce very significantly the computational time of the existing numerical solutions based on an acceptance-rejection procedure for similar performance, and to broaden the class of eligible models for such methods. For instance, in the context of multivariate stochastic differential equations, the proposed algorithm makes use of unbiased estimates of the unknown transition densities under much weaker assumptions than standard alternatives. The performance of this estimator is assessed for high-dimensional discrete-time latent data models, for recursive maximum likelihood estimation in the context of partially observed diffusion process, and in the case of a bi-dimensional partially observed stochastic Lotka-Volterra model.

7.1 Introduction

Latent data models are widely used in time series and sequential data analysis across a wide range of applied science and engineering domains such as movement ecology [194], energy consumptions modelling [37], genomics [297, 96, 282], target tracking [246], enhancement and segmentation of speech and audio signals [228], see also [245, 68, 309] and the numerous references therein. Performing maximum likehood estimation (MLE) for instance with the Expectation Maximization (EM) algorithm [61] or a stochastic gradient ascent ([38] in the case of HMMs)

is a challenging task. Both approaches involve conditional distributions of sequences of hidden states given the observation record (the *smoothing* distribution), which are not available explicitly.

Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC) methods (also known as particle filters or smoothers) are widespread solutions to propose consistent estimators of such distributions. Among SMC methods, algorithms have been designed in the last decades to solve the smoothing problem, such as the Forward Filtering Backward Simulation algorithm [67] or two-filter based approaches [31, 81, 207]. These approaches, which come with strong theoretical guarantees ([59, 67, 71, 100]), require the time horizon and all observations to be available to initialize a backward information filter, and, thus, perform the smoothing. The particle-based rapid incremental smoother [212] is an online version of forward-backward procedures, specifically designed to approximate conditional expectations of additive functionals. This algorithm relies on a backward sampling step performed on the fly thanks to the well known acceptance rejection sampling. This online smoother was proven to be strongly consistent, asymptotically normal, and with a control of the asymptotic variance, when it is performed together with the vanilla bootstrap filter [110]. In [209], the authors show how this algorithm can be used to performed recursive maximum likelihood in state space models. This approach relies on the necessity to upper bound the transition density of the hidden signal, as it is required to perform acceptance rejection sampling.

Moreover, a pivotal step of all SMC approaches is the evaluation of this transition density and of the density of the conditional distribution of an observation given the corresponding latent state (the marginal conditional likelihood). In many practical settings, though, no closed-form expressions of these distributions are available : for instance, in the case of partially observed diffusions [6, 83] or in the context of approximate Bayesian computation smoothing [192]. A first step to bypass this shortcoming was proposed in [80]. The authors proposed an important contribution by showing that it is possible to implement importance sampling and filtering recursions, when the unavailable importance weights are replaced by random estimators. Standard data augmentation schemes were then used to extend this random-weight particle filter to provide new inference procedures for instance for partially observed diffusion models [298].

More recently, the online algorithm of [212] was extended to this setting for partially observed diffusion processes by [102]. Then, [103] introduced a pseudo-marginal online smoother to approximate conditional expectations of additive functionals of the hidden states in a very general setting : the user can only evaluate (possibly biased) approximations of the transition density and of the marginal conditional likelihood. The online algorithm of [103] may be used to approximate expectations of additive functionals under the smoothing distributions by processing the data stream online. However, as with the PaRIS algorihm, when using this pseudo-marginal approach where transition densities are intractable, the user needs to sample exactly from the associated pseudo-marginal backward kernel. This step is again done by rejection sampling, and therefore requires that the estimate of the transition density and of the marginal conditional likelihood are almost surely positive and upper bounded. In practice, these assumptions are very restrictive. For instance, in the context of diffusion processes, they narrow the possible models to the class

of diffusions satisfying the Exact algorithm conditions of [20], for which General Poisson Estimators (GPEs) [79] already lead to eligible unbiased estimators.

In this chapter, a new procedure is introduced to replace the backward acceptance-rejection step of the PaRIS and the pseudo marginal PaRIS algorithms by a backward importance sampling estimate. It leads to a smoothing algorithm that only requires an almost surely positive estimator of the unknown transition or observation density, and therefore extends widely the class of models for which these online smoothers can be designed. In the general case where only signed estimates can be obtained, we propose to use Wald's trick, ensuring positiveness. In the context of partially observed diffusion processes, for instance, we show that combining Wald's trick to the parametrix estimators of [6] and [83] leads to a highly generic algorithm that can be applied to a wide class of models, for which no low variance smoother existed so far.

The paper is organized as follows. Section 7.2 displays the latent data models and the main objectives considered in this chapter. Then, Section 7.3 details online pseudo marginal sequential Monte Carlo algorithms and Section 7.4 our proposed algorithm. Section 7.5 provides extensive numerical experiments to illustrate the performance of our approach. The empirical results of this section can be summarised as follows.

— The proposed approach can be used for any latent data models such as hidden Markov models, or recurrent neural networks with unobserved latent states. Even when the transition densities are available, we show empirically that our backward importance sampling is a computationally efficient solution to solve the online smoothing problem (Section 7.5.1).

— We show that the proposed approach outperforms the existing acceptance rejection method in terms of computational efficiency. (Section 7.5.2).

— We show how the proposed method allows for efficient online recursive maximum likelihood in the context of partially observed diffusion processes (Section 7.5.3).

— When considering the pseudo-marginal approach, we extend the use of Wald's trick to the backward kernel, and therefore show that our approach can be used in cases where the estimators of the unknown densities are not positive by construction.

— We perform sequential Monte Carlo smoothing in models for which no solutions were proposed in the literature to the best of our knowledge, such as multivariate partially observed diffusion processes (Section 7.5.4).

7.2 Model and objectives

Let θ be a parameter lying in a $\Theta \subset \mathbb{R}^q$ and consider a *state space model* where the hidden Markov chain in $\mathbb{R}^d$ is denoted by $(X_k)_{k\geqslant 0}$. The distribution of X_0 has density χ with respect to the Lebesgue measure and for all $0 \leqslant k \leqslant n-1$, the conditional distribution of X_{k+1} given $X_{0:k}$ has density $q_{k+1;\theta}(X_k, \cdot)$, where $a_{u:v}$ is a short-

hand notation for $(a_u, \dots, a_v)$. It is assumed that this state is partially observed through an observation process $(Y_k)_{0 \leqslant k \leqslant n}$ taking values in $\mathbb{R}^m$. For all $0 \leqslant k \leqslant n$, the distribution of Y_k given $X_{0:n}$ depends on X_k only and has density $g_{k;\theta}(X_k, \cdot)$ with respect to the Lebesgue measure. In this context, for any pair of indices $0 \leqslant k_1 \leqslant k_2 \leqslant n$, we define the *joint smoothing distribution* as the conditional law of $X_{k_1:k_2}$ given $Y_{0:n}$. In this framework, the likelihood of the observations $\mathsf{L}_{n,\theta}(Y_{0:n})$, which is in general intractable, is

$$\mathsf{L}_{n,\theta}(Y_{0:n}) = \int \chi(x_0) g_{0;\theta}(x_0, Y_0) \prod_{k=0}^{n-1} \ell_{k;\theta}(x_k, x_{k+1}) \mathrm{d}x_{0:n} \ ,$$

where, for all $0 \leqslant k \leqslant n$ and all $\theta \in \Theta$,

$$\ell_{k;\theta}(x_k, x_{k+1}) = q_{k+1;\theta}(x_k, x_{k+1}) g_{k+1;\theta}(x_{k+1}, Y_{k+1}) \ . \tag{7.1}$$

In a large variety of situations, (7.1) cannot be evaluated pointwise (see models of sections 7.5.2 and 7.5.4), and we assume in this chapter that we have an estimate of this quantity (see assumption **H1** in Section 7.3). Note that to avoid future cumbersome expressions, the dependency of the key quantity $\ell_{k;\theta}(\cdot)$ on the observations is implicit, as we always work conditionnaly to the observations. In this chapter, we propose an algorithm to compute *smoothing expectations of additive functionals*. Namely, we aim at computing $\mathbb{E}[h_{0:n}(X_{0:n})|Y_{0:n}]$, where $h_{0:n}$ is an *additive functional*, *i.e.* a function from $\mathbb{R}^{d \times (n+1)}$ to $\mathbb{R}^{d'}$ satisfying :

$$h_{0:n} : x_{0:n} \mapsto \sum_{k=0}^{n-1} \tilde{h}_k(x_k, x_{k+1}) \ , \tag{7.2}$$

where $\tilde{h}_k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{d'}$. Such expectations are the keystones of many common inference problems in state space models.

Example 1 : *State estimation.* Suppose that the model parameter θ is known, a common objective is to recover the underlying signal X_{k^*} for some index $0 \leqslant k^* \leqslant n$ given the observations $Y_{0:n}$. A standard estimator is $\mathbb{E}[X_{k^*}|Y_{0:n}]$, which is a particular instance of our problem with $\tilde{h}_k(x_k, x_{k+1}) = x_k$ if $k = k^*$ and 0 otherwise.

Example 2 : *EM algorithm.* In the usual case when θ is unknown, the maximum likelihood estimator is given by $\widehat{\theta} = \mathrm{argmax}_{\theta \in \Theta} \ \mathsf{L}_{n,\theta}(Y_{0:n})$. Expectation Maximization based algorithms [61] are appealing solutions to obtain an estimator of $\hat{\theta}$. The pivotal concept of the EM algorithm is that the intermediate quantity defined by

$$\theta \mapsto Q(\theta, \theta') = \mathbb{E}_{\theta'} \left[\sum_{k=0}^{n-1} \log \ell_{k;\theta}(X_k, X_{k+1}) \middle| Y_{0:n} \right]$$

may be used as a surrogate for $\mathsf{L}_n(\theta)$ in the maximization procedure, where $\mathbb{E}_{\theta'}$ is the expectation under the joint distribution of the latent states and the observations when the model is parameterized by θ'. Again, this inference

setting is a special case of our framework where $\tilde{h}_k(x_k, x_{k+1}) = \log \ell_{k;\theta}(x_k, x_{k+1})$.

Example 3 : *Fisher's identity and online gradient ascent.* An alternative to the EM algorithm is to maximize the loglikelihood through gradient based methods. Indeed, in state space models, under some regularity conditions (see [38], Chapter 10), the gradient of the log likelihood can be obtained thanks to Fisher's identity :

$$\nabla_\theta \log \mathsf{L}_n(\theta) = \mathbb{E}_\theta \left[\sum_{k=0}^{n-1} \nabla_\theta \log \ell_{k;\theta}(X_k, X_{k+1}) \middle| Y_{0:n} \right] ,$$

which relies on the expectation of a smoothing additive functional. It has been noted (see [38], chapter 10, or [209] how this identity, coupled with the smoothing recursions of Section 7.3.3, can lead to an online gradient ascent, that provides an online estimate of the MLE. An extension of this method will be illustrated in Section 7.5.3 in a challenging setting where the transition density cannot be evaluated.

7.3 Online sequential Monte Carlo smoothing

7.3.1 Backward statistic for online smoothing

In this section, the parameter θ is dropped from notation for a better clarity. For all pair of integers $0 \leqslant k \leqslant k' \leqslant n$, and all measurable function h on $\mathbb{R}^{d\times(k'-k+1)}$, the expectation with respect to the joint smoothing distribution is denoted by :

$$\phi_{k:k'|n}[h] := \mathbb{E}\left[h(X_{k:k'})|Y_{0:n}\right] .$$

The special case where $k = k' = n$ refers to *filtering distribution* and we write $\phi_k = \phi_{k:k|k}$. A pivotal quantity to estimate $\phi_{0:n|n}[h_{0:n}]$ is the *backward statistic* :

$$\mathbf{T}_k[h_{0:k}](X_k) = \mathbb{E}\left[h_{0:k}(X_{0:k})|X_k, Y_{0:k}\right], \ 1 \leqslant k \leqslant n, \ \mathbf{T}_0 = 0 . \tag{7.3}$$

Note that for each k this statistic is a function of X_k, and is defined relatively to the functional of interest $h_{0:n}$. For additive functionals, this statistic satisfies the two following key identities.

Lemma 2. *For all $n \geqslant 1$ and all $1 \leqslant k \leqslant n$:*

$$\phi_{0:n|n}[h_{0:n}] = \phi_n\left[\mathbf{T}_n[h_{0:n}]\right], \tag{7.4}$$

$$\mathbf{T}_k\left[h_{0:k}\right](X_k) = \mathbb{E}\left[\mathbf{T}_{k-1}\left[h_{0:(k-1)}\right](X_{k-1}) + \tilde{h}_{k-1}\left(X_{k-1}, X_k\right) \middle| X_k, Y_{0:k-1}\right] , \tag{7.5}$$

where $\tilde{h}_{k-1}$ is the function defined in (7.2).

Proof. By (7.3),

$$\mathbf{T}_n[h_{0:n}](X_n) = \mathbb{E}\left[h_{0:n}(X_{0:n})|X_n, Y_{0:n}\right] ,$$

so that $\phi_{0:n|n}[h_{0:n}] = \mathbb{E}[h_{0:n}(X_{0:n})|Y_{0:n}] = \mathbb{E}[\mathbb{E}[h_{0:n}(X_{0:n})|X_n, Y_{0:n}]|Y_{0:n}] = \phi_n[\mathbf{T}_n[h_{0:n}]]$ which concludes the proof of (7.4). On the other hand, for all $1 \leqslant k \leqslant n$, as $X_{0:k-1}$ is independent of Y_k given X_k,

$$\begin{aligned}\mathbf{T}_k\left[h_{0:k}\right](X_k) = \mathbb{E}\left[\sum_{p=0}^{k-1}\tilde{h}_p(X_p, X_{p+1})\middle| Y_{0:k}, X_k\right] &= \mathbb{E}\left[\sum_{p=0}^{k-1}\tilde{h}_p(X_p, X_{p+1})\middle| X_k, Y_{0:k-1}\right] , \\ &= \mathbb{E}\left[h_{0:k-1}(X_{0:k-1})|Y_{0:k-1}, X_k\right] + \mathbb{E}\left[\tilde{h}_{k-1}(X_{k-1}, X_k)\middle| Y_{0:k-1}, X_k\right] .\end{aligned}$$

The proof of (7.5) boils down to using the tower property on the first term of the right hand side. □

Property (7.4) essentially tells us that the target is the filtering expectation of a well chosen statistic, while property (7.4) provides a recursion to compute these statistics. These two properties suggest an *online* procedure to solve the online smoothing problem. Starting at time 0, at each step k, this procedure aims at (i) computing the filtering distribution and (ii) computing the backward statistics. For all $0 \leqslant k \leqslant n-1$, writing $\mathsf{B}_{k-1}(X_k, \cdot)$ the conditional law of X_{k-1} given $(X_k, Y_{1:n})$, Lemma 2 yields

$$\mathbf{T}_k\left[h_{0:k}\right](X_k) = \int \mathsf{B}_{k-1}(X_k, \mathrm{d}x_{k-1})\left\{\mathbf{T}_{k-1}\left[h_{0:k-1}\right](x_{k-1}) + \tilde{h}_{k-1}\left(x_{k-1}, X_k\right)\right\} .$$

As conditionally on $(X_k, Y_{1:n})$, X_{k-1} is independent of $Y_{k:n}$, the backward kernel can be written as follows

$$\mathsf{B}_{k-1}(X_k, \mathrm{d}x_{k-1}) \propto \phi_{k-1}(\mathrm{d}x_{k-1})q_k(x_{k-1}, X_k) = \frac{\phi_{k-1}(\mathrm{d}x_{k-1})q_k(x_{k-1}, X_k)}{\int \phi_{k-1}(\mathrm{d}x_{k-1})q_k(x_{k-1}, X_k)} .$$

Following [79, 211, 102, 103], we do not assume that (7.1) can be evaluated pointwise. We assume that there exists an estimator, relying on some random variable on a general state space $(\mathsf{U}, \mathcal{B}(\mathsf{U}))$ such that the following assumption holds.

H1 For all $\theta \in \Theta$ and $k \geqslant 0$, there exists a Markov kernel on $(\mathbb{R}^d \times \mathbb{R}^d, \mathcal{B}(\mathsf{U}))$ with density $\mathbf{R}_{k;\theta}$ with respect to a reference measure μ on a general state space $(\mathsf{U}, \mathcal{B}(\mathsf{U}))$, and a positive mapping $\ell_{k;\theta}\langle\cdot\rangle$ on $\mathbb{R}^d \times \mathbb{R}^d \times \mathsf{U}$ such that, for all $(x, x') \in \mathbb{R}^d \times \mathbb{R}^d$,

$$\int \mathbf{R}_{k;\theta}(x, x'; z)\ell_{k;\theta}\langle z\rangle(x, x')\mu(\mathrm{d}z) = \ell_{k;\theta}(x, x') .$$

This setup, known as *pseudo marginalisation* is based on the plug-in principle, as a pointwise estimate of $\ell_{k;\theta}(x, x')$ can be obtained by generating ζ from $\mathbf{R}_{k;\theta}(x, x'; \mathrm{d}z)$ and computing the statistic $\ell_{k;\theta}\langle\zeta\rangle(x, x')$. Its use in Monte Carlo methods, and the related theoretical guarantees, have been studied in the context of MCMC [7], and more recently, of SMC [103].

Recursive maximum likelihood

An appealing application for online smoothing is the context of recursive maximum likelihood, i.e., where new observations are used only once to update the estimator of the unknown parameter θ. Following [158], the idea is the build a sequence $\{\theta_k\}_{k\geq 0}$ as follows. First, set the initial value of the parameter estimate : θ_0. Then, for each new observation $Y_k,\ k \geqslant 1$, define

$$\theta_k = \theta_{k-1} + \gamma_k \nabla_\theta \ell_\theta(Y_k \mid Y_{1:k-1}) \ ,$$

where $\ell_\theta(Y_k \mid Y_{1:k-1})$ is the log likelihood for the new observation given all the past, and $\{\gamma_k\}_{k\geqslant}$ are positive step sizes such that $\sum_{k\geqslant 1} \gamma_k = \infty$ and $\sum_{k\geqslant 1} \gamma_k^2 < \infty$. The practical implementation of such an update relies on the following identity :

$$\nabla_\theta \ell_\theta(Y_k \mid Y_{1:k-1}) = \frac{\pi_{k;\theta}[\nabla_\theta g_{k;\theta}] + \eta_{k;\theta}[g_{k;\theta}]}{\pi_{k;\theta}[g_{k;\theta}]} \ , \tag{7.6}$$

where $\pi_{k;\theta} = \phi_{k;\theta|k-1}$ is the predictive distribution and

$$\eta_{k;\theta}[g_{k;\theta}] = \phi_{0:k;\theta|k-1}[h_{0:k;\theta} g_{k;\theta}] - \pi_{k;\theta}[g_{k;\theta}] \times \phi_{0:k;\theta|k-1}[h_{0:k}] \ ,$$

with

$$h_{0:k}(x_{0:k}) = \sum_{j=0}^{k-1} \nabla_\theta \log \ell_{j,\theta}(x_j, x_{j+1}) \ . \tag{7.7}$$

The signed measure $\eta_{k;\theta}$ is known as the *tangent filter*, see [38, Chapter 10], [60] or [209]. Using the tower property and the backward decomposition (7.5) yields

$$\eta_{k;\theta}[g_{k;\theta}] = \pi_{k;\theta}\left[\left(\mathbf{T}_k[h_{0:k}] - \pi_{k;\theta}[\mathbf{T}_k[h_{0:k}]]\right) g_{k;\theta}\right] \ . \tag{7.8}$$

It is worth noting that, in the context of this chapter where ℓ_k cannot be evaluated pointwise, one cannot expect to know the functional (7.7), which involves the gradient of this quantity. In Section 7.5.3, we illustrate that we can plug-in an estimate of this functionnal instead. The rationale motivating this algorithm relies on the following expression of the normalized loglikelihood :

$$\frac{1}{n}\nabla_\theta \ell_\theta(Y_{1:n}) = \frac{1}{n}\sum_{k=1}^{n} \nabla_\theta \ell_\theta\left(Y_k \mid Y_{1:k-1}\right) \ .$$

Moreover, under strong mixing assumptions, for all $\theta \in \theta$, the extended process $\{(X_n, Y_n, \pi_n, \eta_n)\}_{n\geqslant 0}$ is an ergodic Markov chain and for all $\theta \in \theta$, the normalized score $\nabla_\theta \ell_\theta(Y_{1:n})/n$ converges almost surely to a limiting quantity $\lambda(\theta, \theta_\star)$ such that, under identifiability constraints, $\lambda(\theta_\star, \theta_\star) = 0$. A gradient ascent algorithm cannot be designed as the limiting function $\theta \mapsto \lambda(\theta, \theta_\star)$ is not available explicitly. However, solving the equation $\lambda(\theta_\star, \theta_\star) = 0$ may be cast

into the framework of *stochastic approximation* to produce parameter estimates using the *Robbins-Monro algorithm*

$$\theta_k = \theta_{k-1} + \gamma_k \zeta_k \ , \quad n \geqslant 0 \ , \tag{7.9}$$

where ζ_k is a noisy observation of $\lambda(\theta_{k-1}, \theta_\star)$, equal to (7.6). In the case of a finite state space X the algorithm was studied in [158], which also provides assumptions under which the sequence $\{\theta_n\}_{n\geqslant 0}$ converges towards the parameter $\theta_\star$ (see also [270] for refinements).

7.3.2 Approximation of the filtering distribution

Let $(\xi_0^\ell)_{\ell=1}^N$ be independent and identically distributed according to an instrumental proposal density ρ_0 on $\mathbb{R}^d$ and define the importance weights $\omega_0^\ell := \chi(\xi_0^\ell)/\rho_0(\xi_0^\ell)$, where χ is the density of the distribution of X_0 as defined in Section 7.2. For any bounded and measurable function h defined on $\mathbb{R}^d$, the importance sampling estimator defined as

$$\phi_0^N[h] := \Omega_0^{-1} \sum_{\ell=1}^N \omega_0^\ell h(\xi_0^\ell) \ , \quad \text{where} \quad \Omega_0 := \sum_{\ell=1}^N \omega_0^\ell \ .$$

is a consistent estimator of $\phi_0[f]$. Then, for all $k \geqslant 1$, once the observation Y_k is available, *particle filtering* transforms the weighted particle sample $\{(\omega_{k-1}^\ell, \xi_{k-1}^\ell)\}_{\ell=1}^N$ into a new weighted particle sample approximating ϕ_k. This update step is carried through in two steps, *selection* and *mutation*, using sequential importance sampling and resampling steps. New indices and particles $\{(I_k^\ell, \xi_k^\ell, \zeta_k^\ell)\}_{\ell=1}^N$ are simulated independently from the instrumental distribution with density on $\{1, \ldots, N\} \times \mathbb{R}^d \times \mathsf{U}$:

$$v_k(\ell, x, z) \propto \omega_{k-1}^\ell p_{k-1}(\xi_{k-1}^\ell, x) \mathbf{R}_k(\xi_{k-1}^\ell, x; z) \ ,$$

where p_{k-1} is a Markovian transition density. In practice, this step is performed as follows :

1. Sample I_k^ℓ in $\{1, \ldots, N\}$ with probabilities proportional to $\{\omega_{k-1}^j\}_{1\leqslant j\leqslant N}$.
2. Sample ξ_k^ℓ with distribution $p_{k-1}(\xi_{k-1}^{I_k^\ell}, \cdot)$ and sample ζ_k^ℓ with distribution $\mathbf{R}_k(\xi_{k-1}^{I_k^\ell}, \xi_k^\ell; \cdot)$.

For any $\ell \in \{1, \ldots, N\}$, ξ_k^ℓ is associated with the importance weight defined by :

$$\omega_k^\ell := \frac{\ell_{k-1}\langle \zeta_k^\ell \rangle(\xi_{k-1}^{I_k^\ell}, \xi_k^\ell)}{p_{k-1}(\xi_{k-1}^{I_k^\ell}, \xi_k^\ell)} \tag{7.10}$$

to produce the following approximation of $\phi_k[f]$:

$$\phi_k^N[f] := \Omega_k^{-1} \sum_{\ell=1}^N \omega_k^\ell f(\xi_k^\ell) \ , \quad \text{where} \quad \Omega_k := \sum_{\ell=1}^N \omega_k^\ell \ .$$

The choice of the proposal distribution p_{k-1} is a pivotal tuning step to obtain efficient estimations of the filtering distributions. This point will be discussed in each example considered in Section 7.5.

7.3.3 Approximation of the backward statistics

Approximation of the backward statistics, as defined in (7.3), are computed recursively, for each simulated particle. The computation starts with initializing a set $\tau_0^1 = 0, \dots \tau_0^N = 0$, corresponding to the values of $\mathbf{T}_0(\xi_0^1), \dots, \mathbf{T}_0(\xi_0^N)$. The backward kernel is approximated using the particle based estimator of the filtering distribution which yields :

$$\mathsf{B}_k^N(X_{k+1}, \mathrm{d}x_k) = \frac{\sum_{i=1}^N \omega_k^i q_{k+1}(\xi_k^i, X_{k+1}) \delta_{\xi_k^i}(\mathrm{d}x_k)}{\sum_{\ell=1}^N \omega_k^\ell q_{k+1}(\xi_k^\ell, X_{k+1})} .$$

Then, using (7.5), for each $k \geqslant 0$, $1 \leqslant i \leqslant N$, the approximated statistics can be computed as follows

$$\mathbf{T}_{k+1}^N [h_{0:k+1}] (\xi_{k+1}^i) = \frac{\sum_{j=1}^N \omega_k^j q_{k+1}(\xi_k^j, \xi_{k+1}^i) \left\{ \tau_k^j + \tilde{h}_k \left(\xi_k^j, \xi_{k+1}^i \right) \right\}}{\sum_{\ell=1}^N \omega_k^\ell q_{k+1}(\xi_k^\ell, \xi_{k+1}^i)}$$

and updated with :

$$\tau_{k+1}^i = \frac{1}{\widetilde{N}} \sum_{j=1}^{\widetilde{N}} \left(\tau_k^{J_{k+1}^{(i,j)}} + \tilde{h}_k \left(\xi_k^{J_{k+1}^{(i,j)}}, \xi_{k+1}^i \right) \right) , \tag{7.11}$$

where $\widetilde{N} \geqslant 1$ is a sample size which is typically small compared to N and where $(J_{k+1}^{(i,j)}, \zeta_{k+1}^{(i,j)})$, $1 \leqslant j \leqslant \widetilde{N}$, are i.i.d. in $\{1, \dots, N\} \times \mathsf{U}$ with distribution

$$\overline{v}_k^i(\ell, z) \propto \omega_k^\ell \ell_k \langle z \rangle (\xi_k^\ell, \xi_{k+1}^i) \mathbf{R}_k(\xi_k^\ell, \xi_{k+1}^i; z) .$$

As explained in [103], this recursive update requires to produce samples $J_{k+1}^{(i,j)}$ distributed according to the marginal distribution of $\overline{v}_k^i$, referred to as the backward kernel. In practice, this requires computationally intensive sampling procedures and the only proposed practical solution can be used in very restrictive situations. In [102], the authors assumed that almost surely, for all x, x', $\ell_k \langle \zeta \rangle (x, x') \geqslant 0$, and that, for all $0 \leqslant k \leqslant n$ and $0 \leqslant i \leqslant N$, there exists an upper bound $\bar{\varepsilon}_k^i$ such that

$$\sup_{\ell, \zeta} \ell_k \langle \zeta \rangle (\xi_k^\ell, \xi_{k+1}^i) \leqslant \bar{\varepsilon}_k^i . \tag{7.12}$$

Then, if the positiveness assumption of $\ell_k \langle \zeta \rangle (x, x')$ is satistified, the sampling from the distribution $\overline{v}_k$ is possible thanks to condition (7.12), as, for all $(i, z) \in \{1, \dots, N\} \times \mathsf{U}$,

$$\omega_k^\ell \ell_k \langle z \rangle (\xi_k^\ell, \xi_{k+1}^i) \mathbf{R}_k(\xi_k^\ell, \xi_{k+1}^i; z) \leqslant \bar{\varepsilon}_k \omega_k^\ell \mathbf{R}_k(\xi_k^\ell, \xi_{k+1}^i; z) .$$

Therefore, the following accept-reject mechanism algorithm may be used to sample from $\overline{v}_k^i$.

1. A candidate (J^*, ζ^*) is sampled in $\{1, \ldots, N\} \times \mathsf{U}$ as follows :

   (a) J^* is sampled with probabilities proportional to $(\omega_k^\ell)_{\ell=1}^N$;

   (b) ζ^* is sampled independently with distribution $\mathbf{R}_k(\xi_k^{J^*}, \xi_{k+1}^i; \zeta^*)$.

2. (J^*, ζ^*) is accepted with probability $\ell_k\langle\zeta^*\rangle(\xi_k^{J^*}, \xi_{k+1}^i)/\bar{\varepsilon}_k^i$. Upon acceptance, $J_{k+1}^{(i,j)} = J^*$.

This algorithm is the only online SMC smoother proposed in the literature with theoretical guarantees when no closed-form expressions of the transition densities and the conditional likelihood of the observations are available, assuming that the user can only evaluate approximations of these densities. This pseudo-marginal particle smoothing algorithm requires that the backward sampling step generates samples exactly according to $\overline{v}_k^i$. However, it relies on the key assumptions of the positiveness of $\ell_k\langle\zeta\rangle(x, x')$ and (7.12) which are rather restrictive (especially the second one), and would not be satisfied in practice for a lot of problems (see for instance in Section 7.5.4). In Section 7.4, we propose an alternative to this step to obtain a computationally efficient pseudo-marginal smoother in a much wider range of applications for which such assumptions do not hold.

In some cases, bounding the estimator $\ell_k\langle z\rangle(x_k, x_{k+1})$ uniformly in z and x_k is not possible. We may sample from the target distribution using the Metropolis-Hastings algorithm with $\rho_k^i(\ell, z) = \omega_k^\ell \mathbf{R}_k(\xi_k^\ell, \xi_{k+1}^i; z)$ as independent proposal. In this case, $(J_{k+1}^{(i,j)}, \zeta_{k+1}^{(i,j)})_{j=1}^{\widetilde{N}}$ is a Markov chain generated recursively by the following mechanism. Given a state $J_{k+1}^{(i,j)} = J$ and $\zeta_{k+1}^{(i,j)} = \zeta$, a candidate (J^*, ζ^*) for the next state is drawn from ρ_k^i and accepted with probability

$$\alpha := 1 \wedge \frac{\ell_k\langle\zeta^*\rangle(\xi_n^{J^*}, \xi_{n+1}^i)}{\ell_k\langle\zeta\rangle(\xi_n^J, \xi_{n+1}^i)}.$$

In the case of rejection, the next state is assigned the previous state. The resulting Markov chain has the target distribution as stationary distribution and similar to the case of rejection sampling, the acceptance probability can be viewed as a plug-in estimate of the Metropolis-Hastings acceptance probability.

Approximations for recursive MLE

In the case of recursive MLE, one needs to approximate the key quantity (7.6). A particle filter, can be used to compute the following sequential Monte Carlo approximations :

$$\pi_k^N[g_{k;\theta}] = \frac{1}{N}\sum_{\ell=1}^N g_{k;\theta}(\xi_n^\ell) \ , \ \pi_k^N[\nabla_\theta g_{k;\theta}] = \frac{1}{N}\sum_{\ell=1}^N \nabla_\theta g_{k;\theta}(\xi_k^\ell) \ .$$

In addition, the tangent filter can be approximated using a backward sampling procedure, based on the backward statistic associated with the functional (7.7) :

$$\eta_{k;\theta}^N[g_{k;\theta}] = \frac{1}{N}\sum_{\ell=1}^N \tau_\ell^k g_{k;\theta}(\xi_\ell^k) - \left(\frac{1}{N}\sum_{\ell=1}^N \tau_\ell^n\right)\left(\frac{1}{N}\sum_{\ell=1}^N g_{k;\theta}(\xi_\ell^k)\right) \ . \tag{7.13}$$

Plugging these estimates in equation (7.6) allows to perform the online recursive algorithm.

7.4 Pseudo-marginal backward importance sampling

7.4.1 Positive estimates

In this section, we propose to use Wald's identity for martingales to obtain an estimator which is guaranteed to be positive. This step is not required if $\ell_k\langle z\rangle(x,x')$ is positive by construction but this is not necessarily true. Wald's trick was for instance applied in [80] to solve the filtering problem in the context of Poisson based estimators for partially observed diffusions. Our estimator is defined up to an unknown constant of proportionality, which is removed when the importance weights are normalized in equation (7.14). This approach, rather than setting negative weights to 0, which would lead to a biased estimate, uses extra simulation to obtain positiveness. This is done while ensuring that the weights remain unbiased up to a common constant of proportionality.

Particle filtering weights. For all $k \geqslant 0$, the Wald-based random weight particle filtering proceeds as follows.

1. For all $1 \leqslant i \leqslant N$, sample a new particle as described in Section 7.3.2.
   (a) Sample I_k^i in $\{1,\dots,N\}$ with probabilities proportional to $\{\omega_{k-1}^j\}_{1\leqslant j\leqslant N}$.
   (b) Sample ξ_k^i with distribution $p_{k-1}(\xi_{k-1}^{I_k^i},\cdot)$.
2. For all $1 \leqslant i \leqslant N$, set $\omega_k^i = 0$.
3. While there exists $i_* \in \{1,\dots,N\}$ such that $\omega_k^{i_*} \leqslant 0$, for all $1 \leqslant i \leqslant N$, sample ζ_k^i with distribution $\mathbf{R}_k(\xi_{k-1}^{I_k^i},\xi_k^i;\cdot)$ (i.e. compute an estimator of the transition density) and set

$$\omega_k^i = \omega_k^i + \frac{\ell_{k-1}\langle\zeta_k^i\rangle(\xi_{k-1}^{I_k^i},\xi_k^i)}{p_{k-1}(\xi_{k-1}^{I_k^i},\xi_k^i)}\,.$$

We aim at updating the backward statistics τ_{k+1}^i, $1 \leqslant i \leqslant N$ using an importance sampling step as exact accept-reject sampling of the $J_{k+1}^{(i,j)}$, $1 \leqslant j \leqslant \widetilde{N}$, with distribution $\overline{v}_k^i$ requires restrictive assumptions. Therefore, we introduce the following extension of Wald's trick importance sampling to the online smoothing setting of this chapter.

Backward simulation weights. For all $1 \leqslant i \leqslant N$, the backward importance sampling step proceeds then as follows.

1. For all $1 \leqslant j \leqslant \widetilde{N}$, sample $J_{k+1}^{(i,j)}$ in $\{1,\dots,N\}$ with probabilities proportional to $(\omega_k^i)_{i=1}^N$.
2. For all $1 \leqslant j \leqslant \widetilde{N}$, set $\varpi_k^{(i,j)} = 0$.

3. While there exist $j_* \in \{1, \dots, \widetilde{N}\}$ such that $\varpi_k^{(i,j)} \leqslant 0$, for all $1 \leqslant j \leqslant \widetilde{N}$, sample $\zeta_k^{(i,j)}$ with distribution $\mathbf{R}_k(\xi_k^{J_{k+1}^{(i,j)}}, \xi_{k+1}^i; \cdot)$ and set

$$\varpi_k^{(i,j)} = \varpi_k^{(i,j)} + \ell_k\langle\zeta_k^{(i,j)}\rangle(\xi_k^{J_{k+1}^{(i,j)}}, \xi_{k+1}^i) \, .$$

7.4.2 AR-free online smoothing

Without any additional assumption, the statistics are then updated recursively as follows : for all $1 \leqslant i \leqslant N$,

$$\tau_{k+1}^i = \sum_{j=1}^{\widetilde{N}} \frac{\varpi_k^{(i,j)}}{\mathcal{W}_k^i} \left(\tau_k^{J_{k+1}^{(i,j)}} + \tilde{h}_k \left(\xi_k^{J_{k+1}^{(i,j)}}, \xi_{k+1}^i \right) \right) \, , \tag{7.14}$$

where $\varpi_k^{(i,j)}$, $1 \leqslant j \leqslant \widetilde{N}$ are computed using the pseudo-marginal smoothing technique combined with Wald's identity and $\mathcal{W}_k^i = \sum_{j=1}^{\widetilde{N}} \varpi_k^{(i,j)}$. Then, the estimator of the conditional expectation of the additive functional $\phi_{0:n|n}[h_{0:n}]$ is set as

$$\phi_{0:n|n}^{N,\mathrm{IS}}[h_{0:n}] := \sum_{i=1}^{N} \frac{\omega_n^i}{\Omega_n} \tau_n^i \, .$$

7.5 Application to smoothing expectations and score estimation

7.5.1 Recurrent neural networks

Recurrent Neural Networks (RNNs) were first introduced in [201] to model time series using a hidden context state. Such deep learning models are appealing to describe short time dependencies, and RNN extensions [124, 46] are since then widely used in practice, see for instance [195, 264, 265]. In this section, we propose a general state space model based on a vanilla RNN architecture, as follows. The hidden state is initialized as $X_0 \sim \mathcal{N}(0, \Sigma)$ and for all $k \geqslant 1$,

$$X_k = \tanh(W_1 Y_{k-1} + W_2 X_{k-1} + b + \eta_k) \quad \text{and} \quad Y_k = W_3 X_k + c + \varepsilon_k \, ,$$

where W_1, W_2, W_3 and b and c are the weight matrices and bias, Σ is an unknown covariance matrix and $(\eta_k)_{k\geqslant 1}$ and $(\varepsilon_k)_{k\geqslant 1}$ are independent Gaussian random variables with covariance matrices Q and R. In this experiment, we show that the proposed algorithm can be used in this setting which does not fit the usual assumptions of hidden Markov models, as, here, Y_k is not independant from Y_{k-1} conditionnally to the hidden states. While we here focus on a simple vanilla one-layer recurrent network, such general state space model could be extended to multi-layer RNN architectures by considering noisy state dynamics in each hidden layer, and to RNN variants such as Long Short Term Memory [124] and Gated Recurrent Unit (GRU) [46].

To generate a synthetic sequence of states and observations $(X_{0:n}, Y_{0:n})$ from this stochastic RNN, we considered diagonal covariance matrices Σ, Q and R, with the same variance along all dimensions equal to 0.1. To

obtain weights and biases values corresponding to realistic data, we trained a deterministic one-layer RNN on 20,000 samples of a weather time-series dataset available online [1]. In such setting, the observations $Y_{0:n}$ consist in a sequence of 4D vectors (originally temperature, air pressure, air humidity, air density). Following the classical RNN framework, the sequence of hidden states $X_{0:n}$ is usually made of higher-dimensional vectors ; the experiments were performed for two RNNs of respective hidden dimension 32 and 64. After this training part, we sampled $(X_{0:n}, Y_{0:n})$ according to the model. For each RNN, a single sequence of hidden states and observations was simulated with a total length of 200 time steps.

From this general state space model based on a stochastic RNN, in the context of the *state estimation* problem, we are interested in using particle smoothing algorithms to estimate two smoothing expectations, respectively $\mathbb{E}[X_0|Y_{0:n}]$, and $\mathbb{E}[\sum_{k=0}^{n} X_k|Y_{0:n}]$. In our context of online estimation, the evaluation was made for $n = 49$ (sequence truncated at 50 observations), $n = 99$ (sequence truncated at 100 observations), and $n = 199$ (full sequence). The Monte Carlo estimate of these quantities is referred to with the hat symbol : $\widehat{\mathbb{E}}[.]$.

In the following, the performances of our algorithm was compared to the classical Poor Man's smoother. The Poor Man's smoother (also known as the path-space smoother) estimates the joint smoothing distribution using the ancestral lines of each particle at time n ; see for instance [68] for discussions on the path degeneracy issue. For the backward IS smoother, we use $N = 1000$ particles for the bootstrap filter and $\widetilde{N} = 32$ backward samples (see Section 7.5.2 and Figure 7.3 for the choice of $\widetilde{N}$ from the number of particles N). For the Poor Man's smoother $N = 3000$ particles were used, which yields a similar computational cost than the backward IS smoother. One interesting aspect of applying the backward IS smoother (BIS) on neural network architectures is the parallelization abilities of such algorithm : the loop over the backward samples in the backward step of the BIS is easily parallelizable, while the Poor Man's smoother requires to store the full past trajectories of the particles.

Table 7.1 displays the result when performing 100 runs for each smoothing algorithm. The performance metric is the classical mean squared error (MSE), which is approximated with the empirical mean over the 100 runs. The backward IS smoother outperforms the Poor Man's smoother when estimating both quantities. This is also illustrated in Figure 7.1, displaying for the stochastic RNN of dimension 64 the MSE over 100 runs of $\widehat{\mathbb{E}}[X_k|Y_{0:199}]$, for $k \in \{0, ..., 199\}$: the backward IS smoother has a significantly smaller MSE than the Poor Man's smoother for all observations that are recorded far in the past (in our example, for all $k \leqslant 150$). Moreover, the table also shows that while the backward IS MSE tends to stay stable for all given n (49,99, and 199), as expected the Poor Man's estimation is less accurate for a longer sequence of observations, with a MSE increasing as n increases.

7.5.2 One dimensional diffusion processe : the Sine model

This section investigates the performance of the proposed algorithm to compute expectations under the smoothing distributions in a context where alternatives are available for comparison. Consider the Sine model where

1. https ://www.bgc-jena.mpg.de/wetter/

TABLE 7.1 – Empirical estimation of $\mathbb{E}[\|X_0 - \widehat{\mathbb{E}}[X_0|Y_{0:n}]\|^2]$ and $\mathbb{E}[\sum_{k=0}^{n} \|X_k - \widehat{\mathbb{E}}[X_k|Y_{0:n}]\|^2/(n+1)]$ for the Poor man's smoother (PMS) and the Backward IS smoother, for states $X_{0:n}$ and observations $Y_{0:n}$ generated with stochastic RNNs of dimension 32 and 64. We consider an online estimation of a sequence of 200 observations, truncated at timestep $n = 49$ (50 timesteps), $n = 99$ (100 timesteps) and the full sequence ($n = 199$).

	RNN dim = 32		RNN dim = 64	
	PMS	Backward IS	PMS	Backward IS
$\|X_0 - \widehat{\mathbb{E}}[X_0\|Y_{0:199}]\|^2$	0.2147	0.1997	0.1969	0.1649
$\sum_{k=0}^{199} \|X_k - \widehat{\mathbb{E}}[X_k\|Y_{0:199}]\|^2/200$	0.2551	0.2056	0.1822	0.1427
$\|X_0 - \widehat{\mathbb{E}}[X_0\|Y_{0:99}]\|^2$	0.2135	0.1997	0.1914	0.1649
$\sum_{k=0}^{99} \|X_k - \widehat{\mathbb{E}}[X_k\|Y_{0:99}]\|^2/100$	0.2531	0.2189	0.1775	0.1519
$\|X_0 - \widehat{\mathbb{E}}[X_0\|Y_{0:49}]\|^2$	0.2018	0.1997	0.1707	0.1650
$\sum_{k=0}^{49} \|X_k - \widehat{\mathbb{E}}[X_k\|Y_{0:49}]\|^2/50$	0.2307	0.2147	0.1633	0.1589

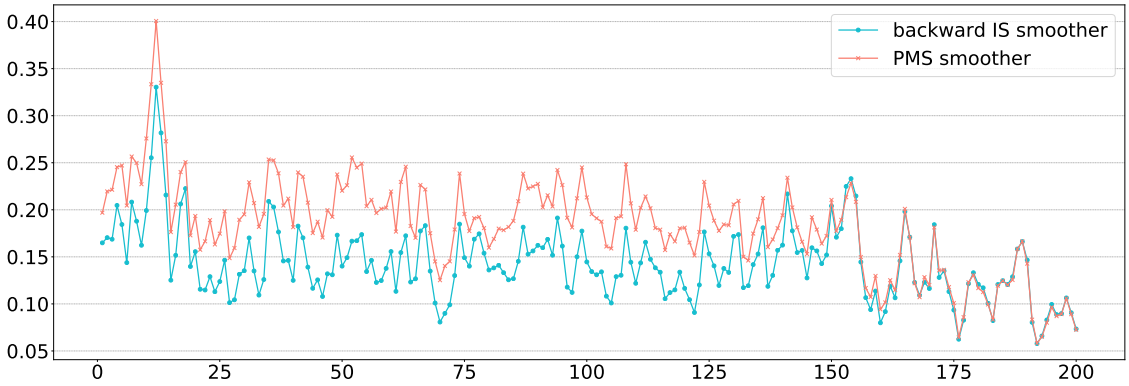


FIGURE 7.1 – Plot of the empirical estimate of $\mathbb{E}[\|X_k - \widehat{\mathbb{E}}[X_k|Y_{0:199}]\|^2]$ for $k \in \{0, ..., 199\}$ for 100 runs of the Backward IS and Poor Man smoothers.

$(X_t)_{t\geqslant 0}$ is assumed to be a weak solution to

$$\mathrm{d}X_t = \sin(X_t - \theta)\mathrm{d}t + \mathrm{d}W_t\ , \quad X_0 = x_0\ .$$

This simple model has no explicit transition density, however, a General Poisson estimator which satisfies (7.12) can be computed by simulating Brownian bridges, (see [21]). Therefore, the backward importance sampling technique proposed in this chapter can be compared to the usual acceptance-rejection algorithm described in Section 7.3.3. For this simple comparison, observations are received at evenly spaced times $t_0 = 0, \ldots, t_{10} = 5$ from the model

$$Y_k = X_{t_k} + \varepsilon_k,\ 0 \leqslant k \leqslant n = 10\ , \tag{7.15}$$

where $(\varepsilon_k)_{0\leqslant k\leqslant 10}$ are i.i.d. Gaussian random variables with mean 0 and variance 1. In this experiment $\theta = \pi/4$. The proposal distribution p_k for the particle filtering approximation is chosen as the following approximation of the

optimal filter :

$$p_k(x_k, x_{k+1}) \propto q_{k+1}^{\mathsf{Eul}}(x_k, x_{k+1}) g_{k+1}(x_{k+1}, Y_{k+1}) \ , \tag{7.16}$$

where q_{k+1}^{Eul} is the probability density function of Gaussian distibution with mean $\Delta \sin(x_k - \theta)$ and variance Δ where $\Delta = 1/2$, i.e. the Euler approximation of the Sine SDE, and g_k is the probability density function of the law of Y_k given X_{t_k} i.e. of a Gaussian random variable with mean X_{t_k} and variance 1. As the observation model is linear and Gaussian, the proposal distribution is therefore Gaussian with explicit mean and variance.

In this first experiment, particles are used to solve the state estimation problem for the first observation i.e. to compute an estimate of $\mathbb{E}[X_0|Y_{0:n}]$. Figure 7.2 displays the computational complexity and the estimation of the posterior mean with the acceptance-rejection algorithm and the proposed backward sampling technique as a function of $\widetilde{N}$. In this setting, $N = 100$, and each unbiased estimate of $\hat{q}$ is computed using 30 Monte Carlo replicates.

For $\widetilde{N} = 2$ (which is the recommended value for the PaRIS algorithm, see [212]), our estimate shows a bias, which is no surprise, as it is based on a biased normalized importance sampling step. However, this bias quickly vanishes for $\widetilde{N} \geqslant 10$. Interestingly, our method comes with a drastic (a factor 10) reduction of computational time. The vanishing of the bias might induce more backward sampling, but this remains much faster than the acceptance rejection method with $\widetilde{N} = 2$.

Then, the same estimation was performed (on the same data set) for N varying from 50 to 2000. In this context, $\widetilde{N}$ was set to 2 for the AR method. To have an empirical intuition of how $\widetilde{N}$ must vary with N for our algorithm, the backward importance sampling is applied with $\widetilde{N} = N^{0.5}, N^{0.6}$ and $N/10$ (as this last value was sufficient in the first experiment to avoid any bias). The results are shown in Figure 7.3. A small bias might appear for $N = 2000$ and $\widetilde{N} = 45$ ($\approx 2000^{0.5}$), but no bias is visible for $N^{0.6}$ and $N/10$. As expected, the gain in time, compared to the state of the art algorithm, remains important (even if it decreases as $\widetilde{N}$ increases). It is worth noting that the variance of the computational time is greatly reduced compared to the AR technique.

7.5.3 Recursive maximum likelihood estimation in the Sine model

Online recursive maximum likelihood using pseudo marginal SMC is illustrated for the same Sine model. As mentionned, a GPE estimator of the transition density can be computed. Following the idea of this computation, it is possible to obtain an unbiased estimate of the gradient of the log-transition density and thus compute and unbiased estimate of the key quantity given in (7.7). To the best of our knowledge, this estimator is new, and given in Appendix C.1. Using the Exact algorithm of [20] a data set of 5000 points (displayed in Figure 7.4), was simulated whith the true parameter $\theta_* = \pi/4$. As in the previous section, particle smoothing was performed, using the same particle filter, with $N = 100$ particles and $\widetilde{N} = 10$ backward samples in our backward importance sampling procedure. In this setup, we explore three key features of our estimator.

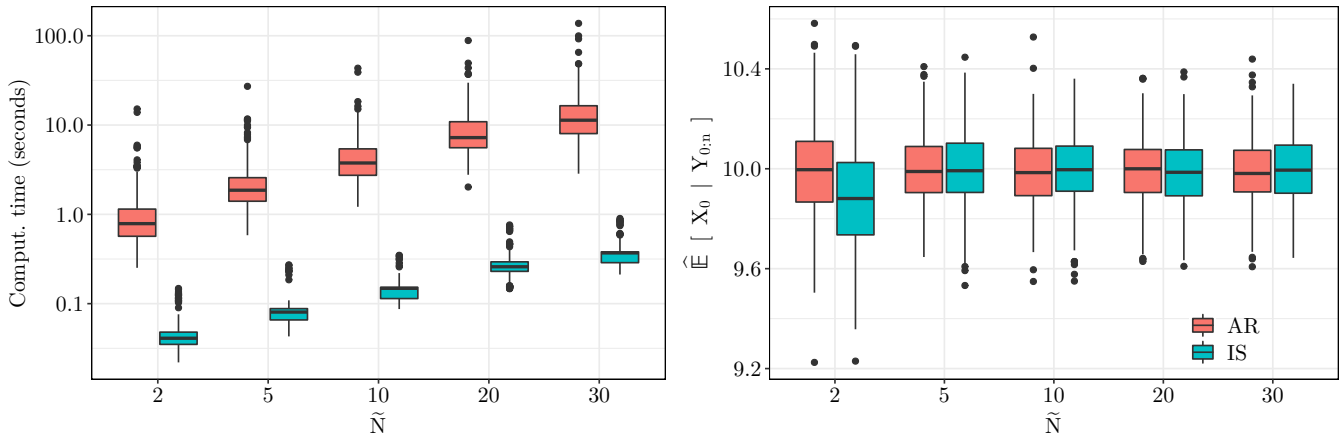


FIGURE 7.2 – Computational complexity and estimation of a posterior mean as a function of the number of backward samples. Results are shown for the state of the art acceptance-rejection (AR) algorithm and the proposed backward importance sampling (IS) technique.

Sensitivity to the starting point $\hat{\theta}_0$. The inference procedure was performed on the same data set from 50 different starting points uniformly chosen in $(0, 2\pi)$. The gradient step size γ_k of equation (7.9) was chosen constant (and equal to 0.5) for the first 300 time steps, and then decreasing with a rate proportional to $k^{-0.6}$. Results are given in Figure 7.4. There is no sensitivity to the starting point of the algorithm, and after a couple of hundred observations, the estimates all concentrate around the true value. As the gradient step size decreases, the estimates stay around the true value following auto-correlated patterns that are common to all trajectories.

Asymptotic normality. The inference procedure was performed on 50 different data sets simulated with the same θ_*. The 50 estimates were obtained starting from the same starting point (fixed to θ_*, as Figure 7.4 shows no sensitivity to the starting point). Figure 7.5 shows the results for the raw and the averaged estimates. The averaged estimates $(\widetilde{\theta}_k)_{k\geqslant 0}$ consist in averaging the values produced by the estimation procedure after a burning phase of n_0 time steps (here $n_0 = 300$ time steps). This procedure allows to obtain an estimator whose convergence rate does not depend on the step sizes chosen by the user, see [225, 151]. For all $0 \leqslant k \leqslant n_0$, $\widetilde{\theta}_k = \widehat{\theta}_k$ and for all $k > n_0$,

$$\widetilde{\theta}_k = \frac{1}{k - n_0} \sum_{j=n_0+1}^{k} \widehat{\theta}_j \ .$$

The estimated distribution of the final estimates seems to be Gaussian, centered around the true value. This conjecture, which would extend the asymptotic normality obtained in [103] for the original pseudo-marginal PaRIS, should be proven in future works.

Step size influence. To illustrate the influence of the gradient step sizes, different settings are considered. In each scenario, the sequence $(\gamma_k)_{k\geqslant 0}$ is given by

$$\gamma_k = \gamma_0 \mathbb{1}_{\{k \leq n_0\}} + \frac{\gamma_0}{(k - n_0)^\kappa} \mathbb{1}_{\{k > n_0\}} \ ,$$

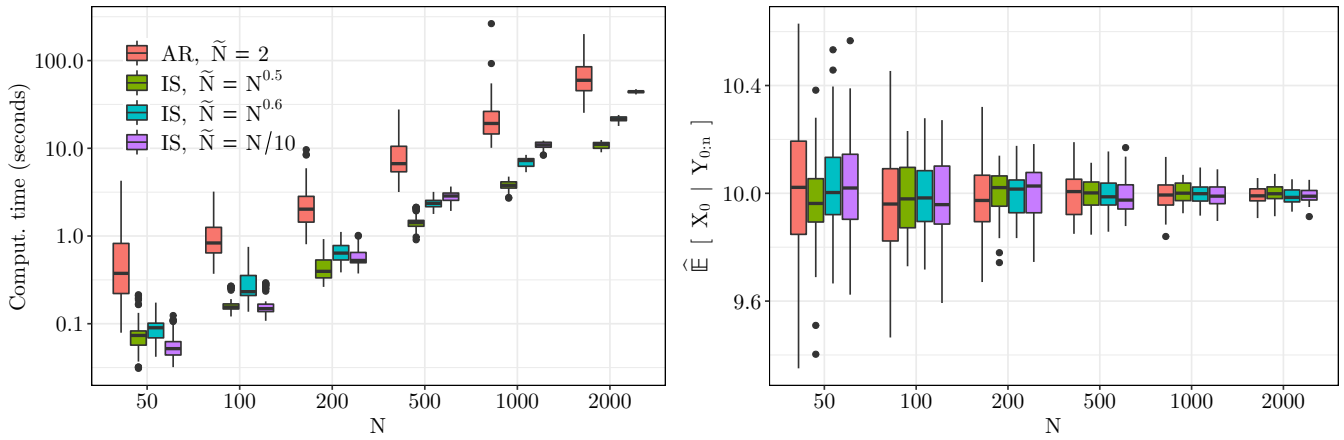


FIGURE 7.3 – Computational complexity and estimation of a posterior mean as a function of the number of particles. Results are shown for the state of the art acceptance-rejection (AR) algorithm and the proposed backward importance sampling (IS) technique. The number of backward samples is set to 2 for the AR, and to $N^{0.5}$, $N^{0.6}$ and $N/10$ for the IS.

where $\gamma_0 = 0.5$. In this experiment $\kappa \in \{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$. The results are shown in Figure 7.6. As expected, the raw estimator shows different rates of convergence depending on κ, whereas the averaged estimator has the same behavior in all cases.

7.5.4 Multidimensional diffusion processes : Stochastic Lotka-Volterra model

This section sets the focus on a stochastic model describing in continuous time the population dynamics in a predator-prey system, as fully discussed in [122]. The bivariate process $(X_t)_{t\geqslant 0}$ of predators and preys abundances is assumed to follow the stochastic Lotka-Volterra model :

$$\mathrm{d}X_t = \alpha_\theta(X_t)\mathrm{d}t + \begin{pmatrix} X_1(t) & 0 \\ 0 & X_2(t) \end{pmatrix} \Gamma \mathrm{d}\mathbf{W}_t \ , \tag{7.17}$$

where $\mathbf{W}_t$ is a vector of independent standard Wiener processes, Γ a 2×2 matrix, and for $x = (x_1, x_2)^T$:

$$\alpha_\theta(x) = \begin{pmatrix} x_1(a_{10} - a_{11}x_1 - a_{12}x_2) \\ x_2(-a_{20} + a_{21}x_1 - a_{22}x_2) \end{pmatrix} .$$

In this context, the unknow parameter to be estimated is $\theta = (a_{10}, a_{11}, a_{12}, a_{20}, a_{21}, a_{22}, \Gamma)$. The observation model follows a widespread framework in ecology where the abundance of preys and predators are observed through

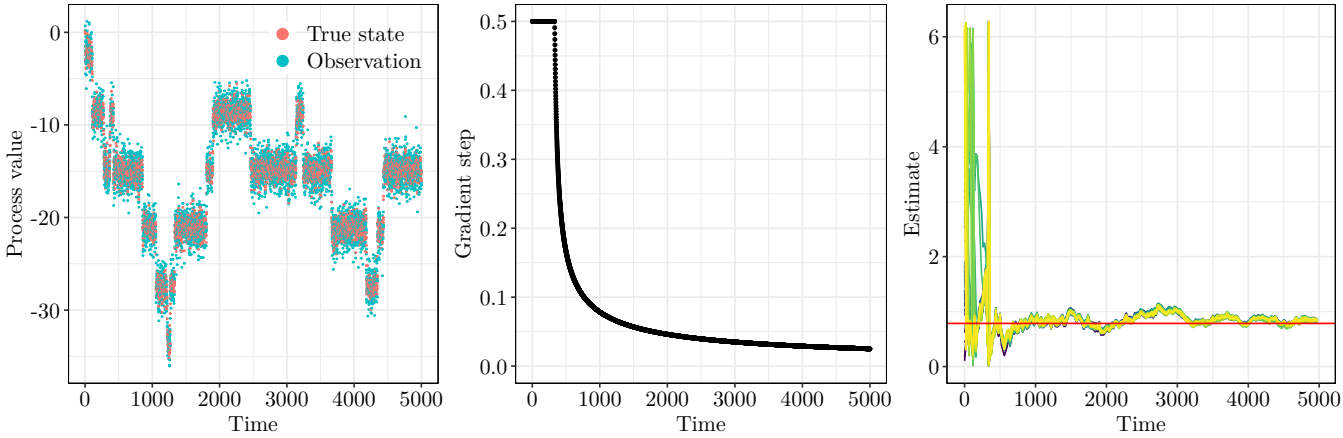


FIGURE 7.4 – (*Left*) Data set simulated according to the SINE process, observed with noise at discrete time steps. (*Middle*) Gradient step sizes (defined in equation (7.9)) for online estimation. (*Right*) Online estimation of θ. Colors differentiate the 50 different starting points. The red horizontal line shows the true value.

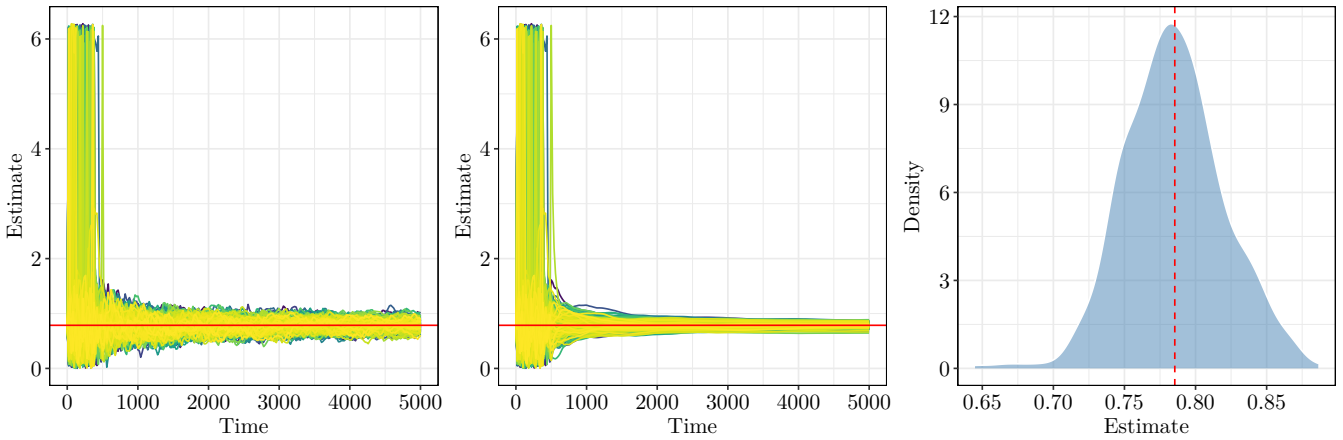


FIGURE 7.5 – (*Left*) online estimation of θ for 50 different simulated data sets. The algorithm is performed from 1 starting point with the gradient step size shown in Figure 7.4. (*Middle*) Averaged estimator, where $\hat{\theta}$ is averaged after a burning phase of 300 time steps. (*Right*) Empirical distribution of $\hat{\theta}$. The red line is the value of θ^*.

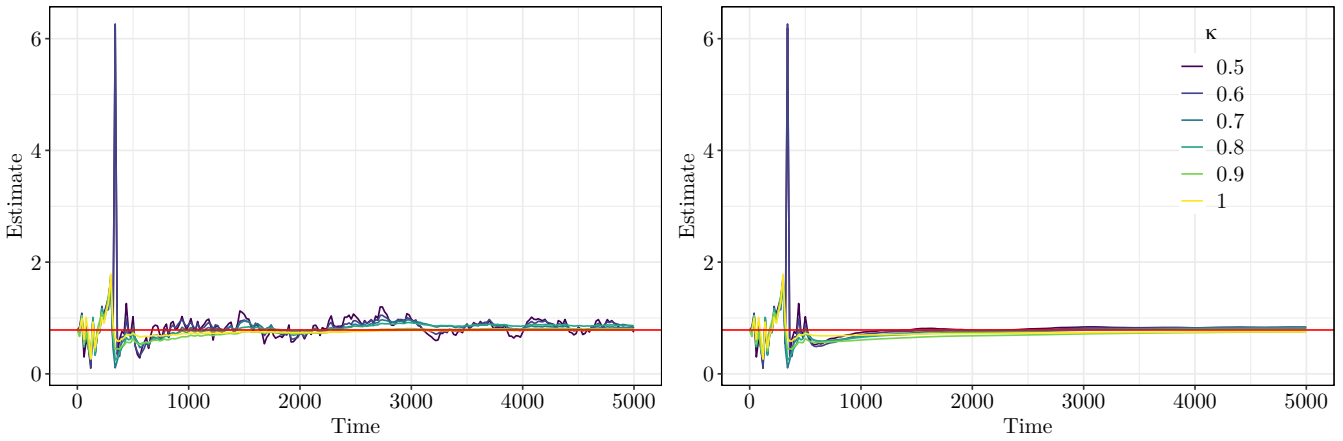


FIGURE 7.6 – (*Left*) online estimation of θ for the data set presented in Figure 7.4, with different decreasing rates values κ. (*Right*) Averaged estimator, where $\hat{\theta}$ is averaged after a burning phase of 300 time steps.

some abundance index at discrete times $t_0, \dots, t_n$:

$$Y_{t_k} = \begin{pmatrix} \mathsf{c}_1 X_1(t_k) \mathrm{e}^{\epsilon_{t_k}^{(1)}} \\ \mathsf{c}_2 X_2(t_k) \mathrm{e}^{\epsilon_{t_k}^{(2)}} \end{pmatrix} , \tag{7.18}$$

where $\mathsf{c} = (\mathsf{c}_1, \mathsf{c}_2)^T$ is known (the observed fraction of the population) and $\{\epsilon_{t_k} = (\epsilon_{t_k}^{(1)}, \epsilon_{t_k}^{(2)})\}_{1 \leqslant k \leqslant n}$ are i.i.d. random variables distributed as a $\mathcal{N}_2(-\text{diag}\ \Sigma/2, \Sigma)$ where Σ is an unknown 2×2 covariance matrix.

Proposal distribution

It is straightforward to show that for a generic θ, in the SDE defined by (7.17), the drift function cannot be written (even after the Lamperti transform) as the gradient of a potential. Therefore, the General Poisson estimator cannot be used as an unbiased estimator of the transition density. Instead, following Section 7.4.1, an almost surely positive unbiased estimate of the transition density is obtained by combining the Wald's trick to the parametrix estimators of [83], as follows. Unlike in the Sine example, it is not straightforward to sample from the p.d.f. proportional to the product

$$q_{k+1}^{\mathsf{Eul}}(x_k, x_{k+1}) g(x_{k+1}) ,$$

as there is no explicit conjugation. A simple Gaussian approximation of the optimal filter is :

$$p_k(x_k, x_{k+1}) \propto q_{k+1}^{\mathsf{Eul}}(x_k, x_{k+1}) \varphi_{y_{k+1}, \Sigma_{\mathsf{prop}}}(x_{k+1}) ,$$

where $\varphi_{y_{k+1}, \Sigma_{\mathsf{prop}}}(x_{k+1})$ the Gaussian p.d.f. whose first moment is the one of $Y_{k+1}|X_{k+1}$ given by (7.18) and variance Σ_{prop} is chosen by the user. We denote $m_{\mathsf{filt}}(\mathbf{x}_k, y_{k+1})$ and Σ_{filt} the mean and variance of this approximate optimal Gaussian filter that can be easily computed. However, particles sampled from this distribution can take negative values, which is not consistent with the model. As a workaround, ξ_{k+1} is defined as $\xi_{k+1} \sim \exp(\varepsilon_{k+1})$ where

$$\varepsilon_{k+1} \sim \mathcal{N}\left(\ln m_{\mathsf{filt}}(\xi_k, y_{k+1}) - \mathsf{diag}(\Sigma_{\mathsf{filt}})/2, \Sigma_{\mathsf{filt}}\right) .$$

The term $\mathsf{diag}(\Sigma_{\mathsf{filt}})/2$ in the mean of ε_{k+1} ensures that $\mathbb{E}\left[\xi_{k+1}\right] = m_{\mathsf{filt}}(\xi_k, y_{k+1})$. In our experiments, Σ_{prop} was chosen as equal to Σ.

Estimation on synthetic data

Simulated data are obtained from the model given by (7.17) and (7.18) for a known set of parameters. Chosen values of θ, Σ, c_1 and c_2 are the following :

$$
\begin{array}{rcll}
a_1 & = & (a_{11} = 12, a_{12} = 0.05, a_{13} = 1) & \text{Prey dynamics parameters} \\
a_2 & = & (a_{21} = 2, a_{22} = 0.2, a_{23} = 0.1) & \text{Predator dynamics param.} \\
\Gamma & = & \begin{pmatrix} 0.5 & 0.1 \\ 0.1 & 0.2 \end{pmatrix} & \text{Diffusion parameters} \\
\mu_0 & = & (50, 20) & \text{Mean of the inital state} \\
\Sigma_0 & = & I_2 & \text{Variance of the initial state} \\
\Sigma_{obs} & = & \begin{pmatrix} 0.01 & 0.005 \\ 0.005 & 0.01 \end{pmatrix} & \text{Observation variance} \\
c & = & (0.2, 0.3) & \text{Detectability values}
\end{array}
$$

The stochastic model is used to simulate abundances indexes $Y_0, \dots Y_{300}$ at times $t_0 = 0, \dots, t_{300} = 3$. The associated time series (after a division by the known constant c) is shown in Figure 7.7 (left panel). In this experiment, the goal is to obtain an estimate of the actual predator-prey abundances given all the observed abundances indexes $Y_{0:n}$. Our estimate is given by the set of conditional expectations $\{\mathbb{E}[X_k|Y_{0:n}]\}_{k=0,\dots,n}$, approximated using our backward importance sampling PaRIS smoother, which is run using the true parameters. Figure 7.7 shows the estimated abundance trajectory over time. The proposed backward importance sampling smoother manages to estimate efficiently the actual abundance from noisy data and a model with an intractable transition density.

Hares and lynx data

In this section, the model defined by equations (7.17) and (7.18) is applied to the Hudson Bay company data, giving the number of hares and lynx trapped in Canada during the first 20 years of the 20th century (available in [208]). As parameters are unknown in this case, maximum likelihood inference is performed using an EM [61] algorithm to obtain an estimate $\hat{\theta}$. The E step is performed using the BIS smoother. At each iteration, the estimator θ_k is updated by finding a parameter θ_{k+1} for which $\hat{Q}(\theta_{k+1}, \theta_k) > \hat{Q}(\theta_k, \theta_k)$, with a gradient free evolution strategy [116]. The last estimate $\hat{\theta}$ obtained with this EM algorithm is used to estimate the actual abundances in the model (similarly to the synthetic data case). Figure 7.8 shows estimates of $\mathbb{E}_{\hat{\theta}}[X_k|Y_{0:n}]$ obtained with 30 independent runs of our algorithm. The particle smoother is implemented using $N = 200$ particles and $\tilde{N} = 20$. The replicates show that the variance of our estimator (for a given set of observations) is much smaller than the one of the Poor Man's smoother.

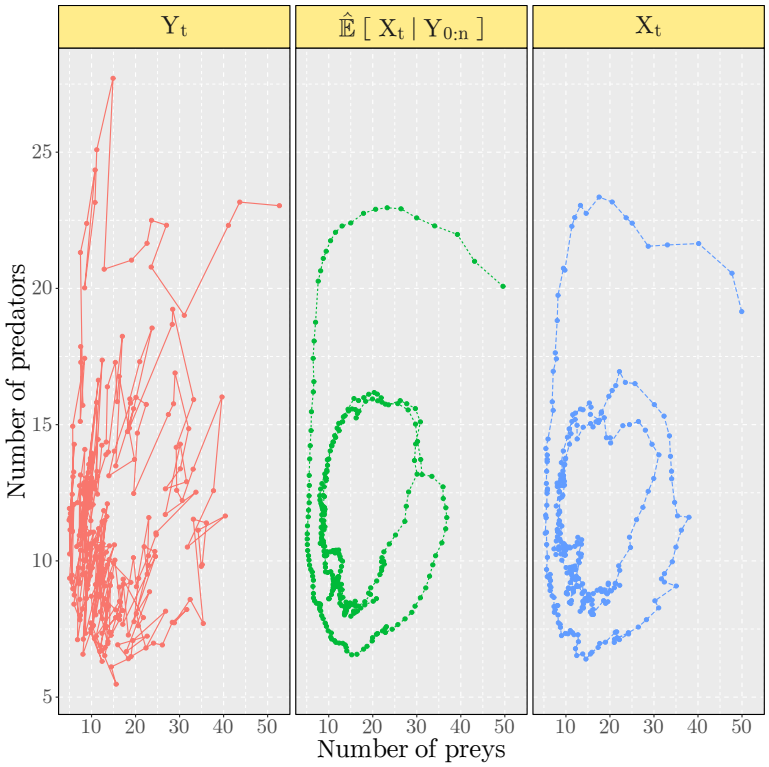


FIGURE 7.7 – Estimated predator-prey abundances (middle) in a stochastic Lotka Volterra model using our backward sampling estimate on simulated abundance indexes (left). Right panel shows the ground truth.

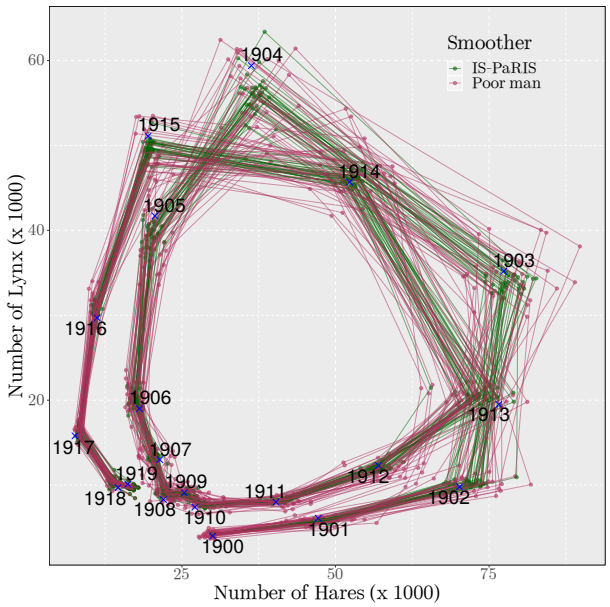


FIGURE 7.8 – Estimated Hares-Lynx abundances using the Hudson bay company data set. Both our IS-PaRIS smoother and the poor man smoother are performed to approximate the MLE and solve the tracking problem. Blue crosses show the observations.

Chapitre 8

Conclusion

8.1 Thesis Summary

In this thesis, we explore new deep learning based approaches for sequential data through the lens of generative modelling using Sequential Monte Carlo methods, and novel reinforcement algorithms for Natural Language Action Spaces. In Chapter 5, we proposed a new RL framework to learn conditional language models by truncating the Vocabulary Action Space with the universal language representation of a pretrained language model. By doing so, we designed one of the first algorithms that enables to learn RL-based language models on large vocabularies from scratch, i.e without requiring supervised datasets. In Chapter 6, we proposed the first deep generative model based on a Transformer network that uses Sequential Monte Carlo Methods to estimate the unobserved latent states and the intractable score function (and therefore to train the model). By doing so, we developed an interesting framework that successfully models uncertainty in time-series forecasting problems, while also showing some potential in fostering language diversity in a language modelling setting. SMC methods are known to be ill-fitted for practical problems using high-dimensional state spaces and based on huge data sets, due to their prohibitive computational costs. With in mind the perspective of scaling these methods for Deep Generative Models, we designed in Chapter 7 a new online smoothing algorithm that (i) has a computational cost much smaller than previous state-of-art approaches, and (ii) is applicable on a wide scope of state-space models, i.e when either the transition density of the latent state or the conditional likelihood of an observation given a state is intractable, which is the case for stochastic neural networks. With the three research works presented within this thesis, we believe to have provided answers to the following questions (that we link with the four research problems presented in section 2.2) :

— *Regarding problem 4 of section 2.2*. RL-based Language Models can be learned from scratch even on complex multi-modal tasks such as Visual Question Generation and on large vocabularies, although it remains a challenge to make these models produce a language that has the same quality as supervised ones.

— *Regarding problem 1 of section 2.2*. Designing stochastic neural networks with complex latent structures

using Sequential Monte Carlo approaches is feasible, and allows to build generative models that perform well on uncertainty quantification tasks on sequential data problems. Yet, using them for solving complex natural language generation tasks remains a challenge.

— *Regarding problem 2 of section 2.2*. We have developed a novel smoothing algorithm whose reduced computational complexity and wider scope of application would make it more suitable for deep learning architectures.

With originally in mind proposing new approaches for representing and learning neural-based language models, we have developed in chapters 6 and 7 algorithms and frameworks that go beyond language modelling, and can be applied to any sequential data problems. One remaining challenge is related to the third research problem mentioned in Section 2.2, i.e building language models which generate both more diverse and high-quality text samples. First solutions have been provided : the TrufLL algorithm from Chapter 5 presents an original language distribution dissimilar from the initial supervised dataset, and the SMC Transformer showcases some abilities to generate diverse text samples when performing sentence continuation. Yet, some work remains to be done to design end-to-end NLG systems that achieve the diversity and quality of language expected to be found in AI systems interacting with humans.

8.2 Future directions

The research works developed within this thesis can be extended in several ways. We list hereafter a few ideas to pursue our works in different directions.

Scaling-up SMC-based Deep Generative Models for NLP problems. The extension of the SMC Transformer to Language Modelling in Section 6.7 has shown that there exists several limits and numerical difficulties when applying the generative model to complex natural language generation tasks and large-scale text datasets. One solution to scale our approach on such settings would be to better involve the addition of noise in the self-attention model, by decoupling the model training phase and testing phase. We could for instance train a deterministic Transformer, and add noise to the model only in a second phase. This would enable to use our approach with pretrained language models, and avoid the computational hurdle and numerical instabilities from adding stochasticity and embedding a SMC algorithm during training. In that case, we could create a plug-and-play top-layer for Language Models with added noise at inference and a SMC algorithm to perform state estimation on the (stochastic) last hidden representation of the Language Model. Intuitively, by perturbing the last hidden representation of the LM with Gaussian noise and considering it as a random variable, this would allow to better explore diverse language possibilities when writing text at inference.

SMC-based approaches for controlled text generation. One closely related problem to fostering language diversity at inference from a trained Language Model, is the issue of controlled text generation. When for example

generating text samples from generic pretrained language models such as the GPT variants [230, 232, 33], we do not have much control over attributes of the output text, such as the topic, the style, the sentiment, etc. Yet, many downstream applications would demand a good control over the model text output. For instance, if we plan to use the LM to generate reading materials for kids, we would like to guide the output stories to be safe, educational and easily understood by children. Recent approaches for solving the controlled text generation problem include (i) designing guided decoding strategies and select desired outputs at test time [101, 126], (ii) optimizing for the most desired outcomes via good prompt design [256], and (iii) fine-tuning the base model or steerable layers to do conditioned content generation [53, 303, 150]. Following the first approach, by combining noise perturbation on the learned hidden representation of the LM with SMC methods, we could design a guided decoding strategy that steers the language generation towards the desired attribute. In that case, the SMC algorithm would perform state estimation on the LM hidden representation, with a resampling mechanism for the particles based on a score function that evaluates the generated text samples according to the desired attribute.

Designing efficient smoothing algorithms for non-markovian settings that handle long-term dependencies. In the state-space model developed in Chapter 6 based on a Transformer network, the dependency between the latent states is non markovian. Indeed, the latent state $X_t = (q_t, \kappa_t, v_t, z_t)$ at timestep t, depends on all the past states $X_{0:t-1}$, plus the current observation Y_t. We use a basic poor man smoother [145] as the smoothing algorithm to estimate the latent states and the score function. Yet, this algorithm suffers from high-variance and from the particles degeneracy phenomena, as shown empirically in Section 6.4.4 ; hence, extending the model and algorithm to more sophisticated smoothing techniques would be a promising improvement. In this perspective, an interesting research problem is the design of forward-backward smoothing algorithms for state-space models with long-term dependencies that handle non-Markovian hidden states.

Extending TrufLL algorithm to the off-policy setting. In Chapter 5, we designed a truncated reinforcement learning algorithm that enables to train a RL-based Language Model from scratch by reducing the effective action space of the RL agent to a smaller set of words. Yet, one limit of our approach mentioned in the Discussion Section 5.5.3 is that we remain in on-policy setting, i.e the policy learned is the truncated policy used to sample episodes during training. In practice, this means that at inference, we need to keep the truncation (and hence the associated pretrained language model) to generate text. One interesting extension would be to design an "off-policy" version of our algorithm, that learns directly a policy over the whole vocabulary, by using during training samples from a behavior policy (the truncated policy). This will (i) speed-up text generation at inference, (ii) allow to generalize to words potentially not selected by the truncation at training, (iii) gets free once the policy is trained from the potential bias present in the pretrained LM, and (iv) consider more flexible RL settings, such as batch reinforcement learning from off-policy data [134]. The discussion section shows that a straightforward application of TrufLL algorithm to

the off-policy setting do not scale to large vocabularies. Yet, there are several improvements possible, for instance plugging TrufLL within sophisticated off-policy policy gradient algorithms such as the Retrace [202], ACER [284], off-PAC [57] algorithms.

Applying the SMC Transformer to active learning settings. In Chapter 6, we demonstrate the ability of the SMC Transformer to provide good predictive uncertainty estimates for sequential data problems, through multiple experiments on time-series forecasting tasks. One interesting use-case using uncertainty estimates is the field of active learning [253, 275], which for a given task focuses the training of a model on a small subset of informative samples, typically samples that are hard to predict, i.e have high uncertainty estimates. In the context of sequence modelling, the SMC Transformer enables to find a posteriori (i.e after training on the full dataset) regions of the dataset space leading to high uncertainty, that could be used, in a second phase, in an active learning setting to improve the model overall predictive performance. This is particularly interesting in a reinforcement learning framework, where episodes sampled during training can have a high impact on the policy being learned. For instance, in Chapter 5, on visual question generation tasks, the learned language model policy from TrufLL suffers to some extent from mode collapse, i.e on "difficult" (image, answer) pairs, the RL agent outputs a generic question that he might have sampled a lot through training. Having a measure that quantifies the uncertainty of such "hard to learn" samples would allow to detect them automatically, and fine-tune in a second phase the RL agent by focusing on these samples. Additionally, one challenge when building RL-based AI systems is the concept of safe reinforcement learning [188], i.e RL with uncertainty-aware estimates that allow to avoid fatal failure cases for satefy-critical tasks. In such settings, combining generative modelling with RL would allow to detect out-of-distribution samples at test time and mitigate risky behaviors from the RL agent.

Annexe A

Appendix for Chapter 5

A.1 training details and hyper-parameters search

We optimize the full loss $L = L_{PPO} + \alpha L_{VF} + \beta L_E$ with $\alpha = 0.5$, $\beta = 0.01$ and a PPO clipping ratio $\epsilon = 0.02$ (resp. 0.01) for CLEVR (resp. VQAv2). We use Adam optimizer [141] with a learning rate (lr) of 10^{-3} for TrufLL and the scratch baseline, 10^{-5} (resp. 10^{-6}) for RL algorithms with a pre-training phase on CLEVR (resp. VQAv2), and $5 * 10^{-4}$ for models including a KL regularization term. We use a batch size (bs) of 128 for all models except the ones with KL regularization, for which we use a batch size of 64. Finally, for the RL from scratch baselines, we perform gradient clipping (gladclip) of 1 (resp. 5) for CLEVR and VQAv2.

Such hyper-parameters were selected, after conducting an extensive hyper-parameter search. The following values were tested : $\beta \in \{0.01, 0.02, 0.05, 0.1\}$, $\epsilon \in \{0.01, 0.02, 0.05, 0.1, 0.5, 0.9\}$, lr $\in \{10^{-6}, 10^{-5}, 10^{-4}, 5*10^{-4}, 10^{-3}, 5*10^{-3}, 10^{-2}, 5*10^{-2}\}$, gradclip $\in \{\text{None}, 1, 5, 10, 100\}$, bs $\in \{32, 64, 128\}$.

Additionally, we also tested for VQAv2 policy networks with 64, 256 and 1024 units, with respectively 32, 128 and 512 word embedding dimensions. We kept the network size giving the best performances, i.e. policy network of 256 units and 128 word embedding dimension.

A.2 Additional experiments

A.2.1 CLEVR

Table A.1 displays the complete ablation on the truncation functions with parameters sweep. The 'sizeVA' variable indicates the average size of the truncated action space for each truncation function. Table A.2 displays the ablation over the three decoding procedures defined in Section 5.4.5. Such an ablation presents a similar pattern than VQAv2 results described in section 5.5.2.

Finally, Table A.3 reports CLEVR metrics when using the BLEU score as the reward. While on such a task TrufLL still exhibits promising language scores, the n-grams metrics remain lower than the pretrained baselines. This illustrates that using a language similarity score as a reward signal is much less interesting than a reward based on a task completion score.

TABLE A.1 – **CLEVR task :** Ablation on the truncation functions with parameters sweep. Best values are in bold.

trunc.	Score	BLEU	CIDEr	ppl-e(↓)	sBLEU(↓)	Size VA
TrufLL (Task-LM)						
$\mathrm{top(k=10)}$	0.50	0.12	0.32	10^2	0.93	10
$\mathrm{top(k=20)}$	0.45	0.10	0.24	10^3	0.87	20
$\mathrm{p_{th}}(\alpha=0.05)$	**0.55**	**0.18**	0.63	25	0.96	4.4
$\mathrm{p_{th}}(\alpha=0.1)$	0.47	0.18	**0.87**	6.7	0.98	2.4
$\mathrm{p_{th}}(\alpha=1/\mathrm{V})$	0.50	0.16	0.49	41	0.97	6.6
$\mathrm{top(p=0.85)}$	0.52	0.17	0.69	10.4	0.96	4.6
$\mathrm{top(p=0.9)}$	0.51	0.17	0.69	11.5	0.96	5.1
$\mathrm{sample(k=20)}$	0.50	0.18	0.73	18.9	**0.86**	5.4
$\mathrm{sample(k=30)}$	0.50	0.18	0.73	16.1	0.89	6.1
TrufLL (Ext-LM)						
$\mathrm{top(k=10)}$	0.52	0.06	0.15	10^2	0.94	10
$\mathrm{top(k=20)}$	0.48	0.05	0.12	10^2	0.89	20
$\mathrm{p_{th}}(\alpha=0.05)$	0.48	0.08	0.34	3.03	0.95	3.3
$\mathrm{p_{th}}(\alpha=0.1)$	0.45	0.17	0.74	**2.2**	0.99	2.1
$\mathrm{p_{th}}(\alpha=1/\mathrm{V})$	0.44	0.11	0.37	3.7	0.96	5.7
$\mathrm{top(p=0.85)}$	0.45	0.10	0.39	3.2	0.92	4.1
$\mathrm{top(p=0.9)}$	0.48	0.15	0.57	2.8	0.97	4.3
$\mathrm{sample(k=20)}$	0.45	0.14	0.50	2.4	0.92	4.1
$\mathrm{sample(k=30)}$	0.43	0.13	0.46	2.7	0.92	4.6

TABLE A.2 – **CLEVR task :** Ablation on sampling methods. Best overall values are underlined, while best values for TruFLL are in bold.

method	text-gen	score	BLEU	CIDEr	ppl-e
pretrain	greedy	0.32	0.22	1.01	14
	sampling	0.29	0.17	0.76	58
	lm-ranking	0.28	0.18	0.73	20
pretrain + RL	greedy	0.53	0.18	0.73	24
	sampling	0.40	0.16	0.68	39
	lm-ranking	0.40	0.17	0.68	5
Task-LM	greedy	**0.57**	0.17	0.65	39
	sampling	0.55	**0.17**	**0.66**	24
	lm-ranking	0.51	0.16	0.65	9
Ext-LM	greedy	0.48	0.09	0.34($\pm$0.11)	3.0
	sampling	0.48	0.10	0.35($\pm$0.11)	3.1
	lm-ranking	0.48	0.06	0.34($\pm$0.11)	**2.9**

A.2.2 VQAv2

Temperature scheduling : On the CLEVR task, we observed that dynamic truncations outperform static ones such as $\mathrm{top(k)}$: indeed, they better take into account the inherent variability of the language structure at the sentence-level. When scaling up to the 15k words of the VQAv2 task, we also dynamically decrease the truncation

TABLE A.3 – **CLEVR, BLEU reward**. Scores are averaged over the three decoding procedures detailed in Section 5.4.5 and over 5 seeds, standard deviation are displayed whenever greater than 0.01 for accuracy metrics. We here report the models with the highest task-success, i.e. the *scratch with KL regularization* baseline with $\lambda_{KL} = 0.1$, and the truncation model with a probability threshold, $\mathrm{p_{th}}(\alpha = 0.05)$. Baseline and Metrics are respectively detailed in Section 5.4.4 and 5.4.3. Best overall values are underlined, while best values for models without task-data (i.e RL from scratch algorithms) are in bold.

Method	Score	R@5	BLEU	Meteor	CIDEr	ppl-t (↓)	ppl-e (↓)	sBLEU (↓)	peak.(↓)
pretrain	0.30	0.71	0.19	0.38	0.83	3.1	31	0.44	0.96
pretrain + RL fine-tune	0.34	0.80	0.20	0.38	0.83	3.8	12	0.56	0.96
scratch	0.03	0.19	0.06	0.09	0.09	10^8	10^6	**0.13**	**0.14**
scratch + KL-task	0.09	0.33($\pm$0.15)	0.15	0.31	0.58($\pm$0.23)	3.8	63	0.34	0.95
scratch + KL-ext	0.06	0.30($\pm$0.23)	0.13	0.25	0.42	10^3	3.6	0.37	0.96
scratch + Truncation-task	**0.17**	**0.51**	**0.18**	**0.37**	**0.80**	**2.6**	17	0.63	1.0
scratch + Truncation-ext	0.07	0.36	0.16	0.29	0.49	10^2	**2.3**	0.60	1.0

size through training, by applying a decreasing temperature schedule on the language model. While temperature scaling [11] is usually used at test time to control the smoothness of the language model distribution, temperature schedules during training of language models have been used in several settings [131, 304, 281]. Formally, $f_{LM}(w_i|w_{<t})$ distribution is computed as $\mathrm{softmax}(x_i) = e^{-x_i/\tau} / \sum_j e^{-x_j/\tau}$, with x_j the LM logits and τ the temperature, which decreases from τ_{max} to τ_{min} by a factor T_F every T_u training step. In Table A.4, both TrufLL (Task-LM) and TrufLL (Ext-LM) benefit slightly from truncation with a temperature schedule compared to a vanilla truncation. The former displays the best performance/language scores trade-off for the schedule "τ : 3 > 1. & T_u=5,000", while the latter has the best metrics trade-off for "τ : 1.5 > 1. & T_u=5,000".

Finally, Figure A.1 displays the evolution of the training return for TrufLL and the baselines. As expected, the pretrain+RL fine-tune baseline return does not evolve much, confirming that the policy distribution almost does not shift through the fine-tuning phase. The training curves of TrufLL present a steady increase in the return until reaching convergence, confirming that our approach, by guiding the exploration of the action space, provides a sufficient learning signal. On the other hand, the scratch+KL baselines stay stuck to a low training return. This suggests that the KL regularization term, while encouraging the policy distribution to resemble the language model distribution, fails to capture the task pragmatics, which requires generating a language that is visually grounded.

TABLE A.4 – **VQA task :** Ablation on the temperature schedules. "no temp. sch" is a classic truncation without temperature scheduling. We then report different schedules $\tau : \tau_{max} > \tau_{min}, T_u$, with τ_{max}, τ_{min}, T_u, and $T_f = 0.75$ as defined in section A.2.2. Best values are in bold.

Scheduling		Score	BLEU	CIDEr	ppl-e(↓)	sBLEU(↓)
TrufLL (Task-LM)						
no temp. sch		**0.35**	0.20	0.11	10^2	0.78
τ : 1.5 > 1.	T_u=5,000	0.34	0.18	0.11	10^2	0.79
τ : 3 > 1.	T_u=5,000	**0.35**	0.22	0.13	10^2	0.76
τ : 1.5 > 1.	T_u=15,000	0.31	**0.23**	**0.23**	10^2	**0.73**
TrufLL (Ext-LM)						
no temp. sch		0.34	0.18	0.04	25	0.83
τ : 1.5 > 1.	T_u=5,000	0.33	0.19	0.05	**20**	0.83
τ : 3 > 1.	T_u=5,000	0.32	0.15	0.05	35	0.82
τ : 1.5 > 1.	T_u=15,000	0.29	0.16	0.08	38	0.68

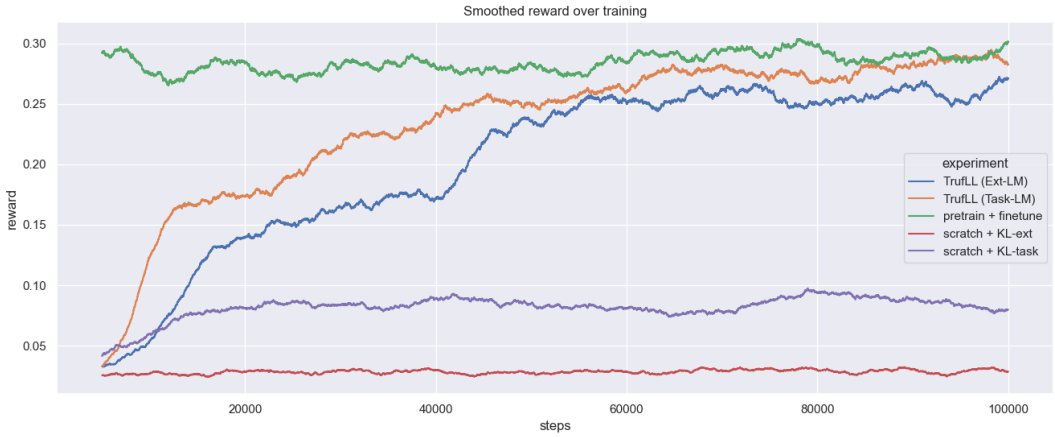


FIGURE A.1 – **VQAv2 :** Training curves. Reward is a rolling average over 5000 timesteps.

A.2.3 Additional discussion

TrufLL with a pre-training phase. Although TrufLL aims at providing a robust method to learn a language model (almost) from scratch, we investigate whether such algorithm can be complementary to RL algorithms with a pre-training phase. Therefore, when using the task-related dataset, we evaluate TrufLL from a pretrained policy, and we refer to it as $\text{TrufLL}_{\text{pretrain}}$.

In table A.5, while on CLEVR, $\text{TrufLL}_{\text{pretrain}}$ marginally improves the results of the pretrain+RL fine-tune baseline, the combination of TrufLL with a pre-training phase leads to performance degradation on VQAv2. This suggests that on a large vocabulary task, the language distribution learned by the SL pretrained policy is significantly different from the one learned with TrufLL.

TABLE A.5 – $\text{TrufLL}_{\text{pretrain}}$ results on the 2 tasks. Additionally, we report the results for the pretrain+RL fine-tune baseline as a comparison. Best values are in bold.

Algo	Score	BLEU	CIDEr	ppl-e	sBLEU
CLEVR					
pretrain+RL	0.44	0.17	0.70	35	**0.46**
$\text{TrufLL}_{\text{pretrain}}$	**0.61**	**0.18**	**0.77**	**22**	0.84
VQAv2					
pretrain+RL	**0.41**	**0.31**	**0.98**	50	**0.78**
$\text{TrufLL}_{\text{pretrain}}$	0.33	0.27	0.42	**35**	1.0

On-policy TrufLL versus off-policy TrufLL. To ease off-policy learning, we propose to add a KL-regularization term in the RL loss [132, 133, 290], and refer to it as $\text{TrufLL}_{\text{off,KL}}$. Intuitively, it encourages the policy to stay close to the language model's distribution, with a distribution support attributing negligible probabilities to words outside the truncated action space.

Table A.6 displays the full results of on-policy versus off-policy scores for TrufLL (Task-LM) and TrufLL (Ext-LM) on the two tasks. The full results emphasize the challenges of the approach for the large vocabulary of VQAv2. Indeed, on the off-policy setting for such a task, the exploding values for e-ppl suggest that the optimized language agent samples incoherent words taken outside the truncated action space, as corroborated by the low values of the sumVA ratio.

Interestingly, while on CLEVR, $\mathrm{TrufLL}_{\mathrm{off,KL}}$ trades off task performance for language quality when compared to $\mathrm{TrufLL}_{\mathrm{off}}$, on VQAv2, it mainly provides a better learning signal for the complete (large) vocabulary. In such a setting, it hence improves the global scores of the off-policy version of TrufLL, and enables a much better generalization at test time of the global policy over the full vocabulary. Yet, keeping truncation at test time remains crucial with large vocabulary. Note that for VQAv2, the poor performances of $\mathrm{TrufLL}_{\mathrm{off,KL}}$ on the external LM is mainly due to numerical instability challenges when using GPT-2 as the target policy of the KL regularization term.

Additionally, on-policy versus off-policy scores split per sampling procedure are displayed in table A.7 : unsurprisingly, greedy decoding for $\mathrm{TrufLL}_{\mathrm{off}}$ outperforms the two sampling-based methods, that are more penalized by the imperfect generalization of the optimized policy over the full vocabulary.

TABLE A.6 – On-policy vs. off-policy scores for different variants of TrufLL : when training with an off-policy loss, we remove the truncation at test time. $\mathrm{TrufLL}_{\mathrm{off,KL}}$ is evaluated with $\lambda_{KL} = 0.05$. Best values are in bold.

Algo	Score	BLEU	CIDEr	ppl-e	sBLEU	sumVA
CLEVR						
TrufLL (Task-LM)						
TrufLL	**0.56**	**0.17**	0.06	10^3	0.78	N.A
$\mathrm{TrufLL}_{\mathrm{off}}$	0.50	0.14	0.43	10^4	0.88	0.96
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	0.39	0.17	**0.71**	69	**0.48**	0.95
TrufLL (Ext-LM)						
TrufLL	0.48	0.08	0.34	**3.03**	0.95	N.A
$\mathrm{TrufLL}_{\mathrm{off}}$	0.41	0.10	0.35	10^5	0.88	0.95
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	0.35	0.15	0.60	20	0.55	0.96
VQAv2						
TrufLL (Task-LM)						
TrufLL	**0.35**	0.21	0.11	10^4	0.36	N.A
$\mathrm{TrufLL}_{\mathrm{off}}$	0.07	0.03	0.01	10^4	0.05	0.08
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	0.12	**0.24**	**0.25**	10^3	0.26	0.71
TrufLL (Ext-LM)						
TrufLL	0.34	0.18	0.04	**24**	0.83	N.A
$\mathrm{TrufLL}_{\mathrm{off}}$	0.09	0.04	0.01	10^4	**0.05**	0.07
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	0.0	0.15	0.02	10^3	0.19	0.47

A.3 Human Evaluation details

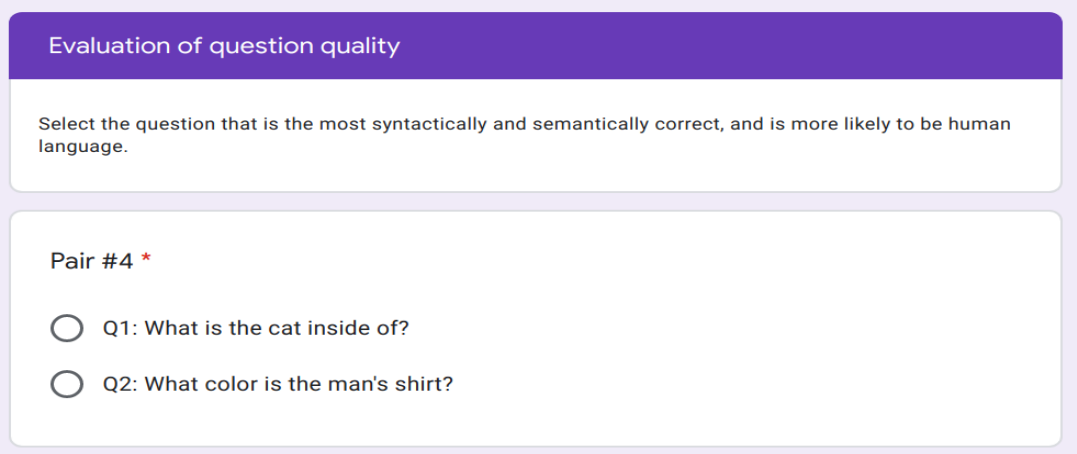
For the Human Evaluation study, we designed one form per participant, with three sections evaluating respectively the language quality, language grounding and diversity criteria. Given the five evaluated models, there are ten

TABLE A.7 – On-policy vs. off-policy scores per decoding procedure : when training with an off-policy loss, we remove the truncation at test time. $\mathrm{TrufLL}_{\mathrm{off,KL}}$ is evaluated with $\lambda_{KL} = 0.05$. Best values are in bold.

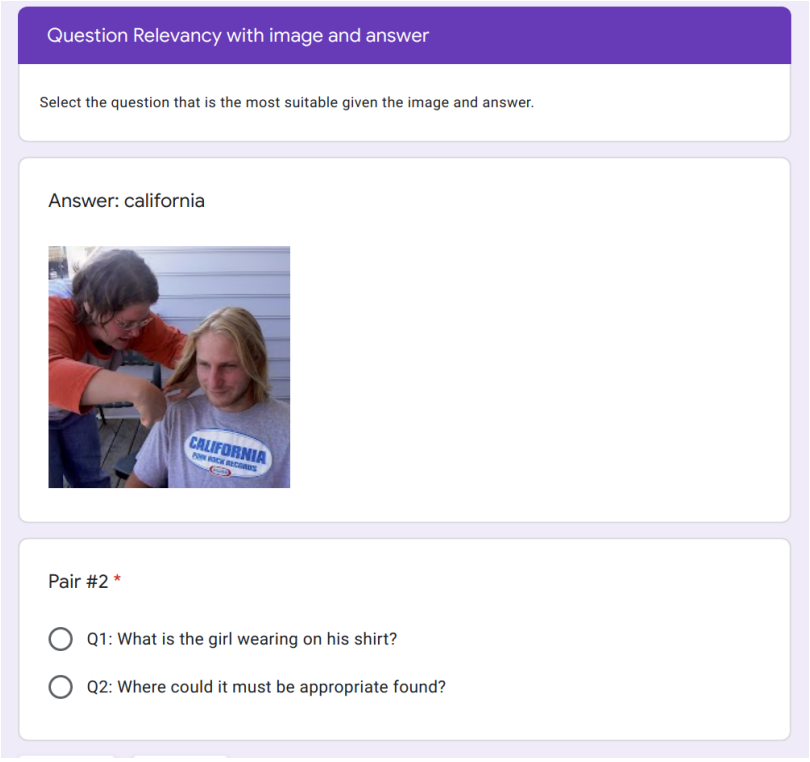
method	text-gen	score	BLEU	CIDEr	e-ppl
CLEVR					
TrufLL (Task-LM)					
TrufLL	greedy	**0.57**	0.17	0.65	39
	sampling	0.55	0.17	0.66	24
	lm-ranking	0.51	0.16	0.65	8.8
$\mathrm{TrufLL}_{\mathrm{off}}$	greedy	0.52	0.17	0.58	71
	sampling	0.49	0.16	0.59	10^5
	lm-ranking	0.48	0.17	0.58	19
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	greedy	0.56	**0.18**	**0.78**	24
	sampling	0.31	0.16	0.62	10^2
	lm-ranking	0.31	0.18	0.74	5.8
TrufLL (Ext-LM)					
TrufLL	greedy	0.48	0.09	0.34	3.1
	sampling	0.48	0.10	0.35	3.1
	lm-ranking	0.48	0.06	0.34	2.9
$\mathrm{TrufLL}_{\mathrm{off}}$	greedy	0.42	0.10	0.38	4.4
	sampling	0.40	0.10	0.35	10^6
	lm-ranking	0.40	0.10	0.34	15
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	greedy	0.48	0.16	0.70	2.1
	sampling	0.27	0.13	0.48	55
	lm-ranking	0.30	0.16	0.61	**2.0**
VQAv2					
TrufLL (Task-LM)					
TrufLL	greedy	0.36	0.20	0.11	366
	sampling	0.35	0.20	0.11	337
	lm-ranking	0.34	0.21	0.11	95
$\mathrm{TrufLL}_{\mathrm{off}}$	greedy	0.09	0.04	0.02	10^3
	sampling	0.05	0.03	0.01	10^6
	lm-ranking	0.06	0.03	0.01	10^4
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	greedy	0.16	0.29	0.46	38
	sampling	0.08	0.19	0.09	10^4
	lm-ranking	0.12	0.24	0.22	10^2
TrufLL (Ext-LM)					
TrufLL	greedy	**0.48**	0.09	0.34	3.1
	sampling	0.48	0.10	**0.35**	3.1
	lm-ranking	0.48	0.06	0.34	**2.9**
$\mathrm{TrufLL}_{\mathrm{off}}$	greedy	0.11	0.05	0.01	10^2
	sampling	0.07	0.03	0.01	10^5
	lm-ranking	0.08	0.04	0.01	10^4
$\mathrm{TrufLL}_{\mathrm{off,KL}}$	greedy	0.00	**0.18**	0.05	27
	sampling	0.00	0.13	0.01	10^3
	lm-ranking	0.00	0.16	0.02	10^2

different model pairs : each section of the form contains 10 pairwise comparison covering all the possible model pairs for the criteria. Each pairwise comparison is sampled uniformly over the 50 first question samples generated by the algorithms at test time. The evaluation of syntax errors was made within the diversity section : for each questions pair, we asked participants to tick the questions if they are grammatically incorrect. Figure A.2 displays one pairwise comparison example for the three sections, and a full form example is available at the following url :

https://forms.gle/kkL38x31wF7A9YKx5.



(a) Language Quality pairwise comparison

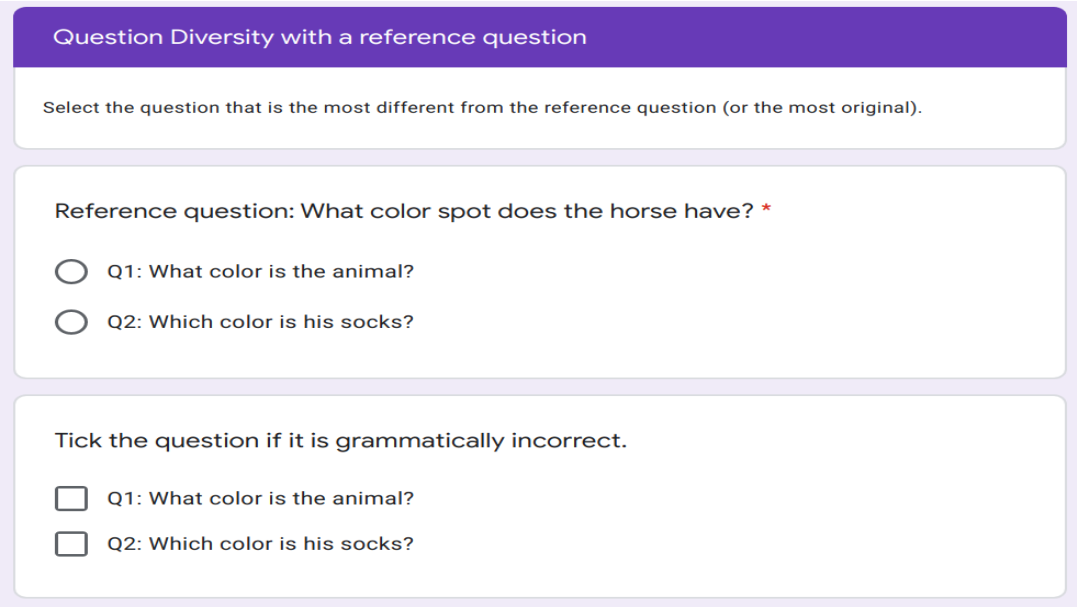


(b) Language Grounding pairwise comparison

FIGURE A.2 – Examples of pairwise comparison for each evaluated criteria.

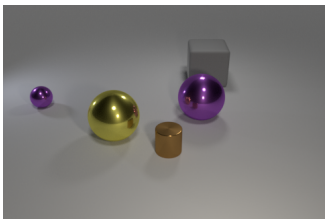
A.4 Additional VQG Samples

Figure A.3 and Figure A.4 display the 10 first dialog samples produced at test time on CLEVR, while figures A.5, A.6, and A.7 display the 15 first dialog samples produced at test time on VQAv2.

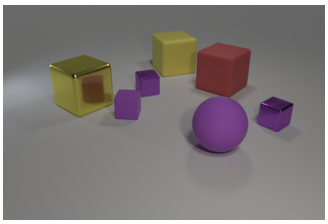


(a) Diversity/Originality with reference question. Pairwise comparison and evaluation of syntax errors.

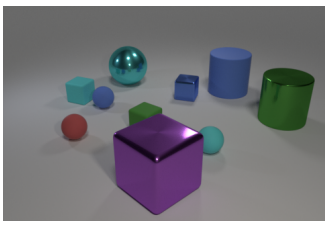
FIGURE A.2 – Examples of pairwise comparison for each evaluated criteria. (cont.)



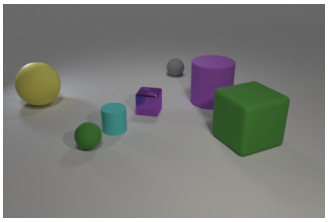
Human	the big yellow object is what shape ? **A :sphere**
pretrain	there is a small thing that is the same color as the small rubber cylinder ; what is its shape ?
pretrain + RL	What is on the person's head ?
scratch	yellow on or an material ?
scratch+KL-task	what number of other things are the same shape as the small gray thing ?
scratch+KL-ext	does that tiny object have objects to its left ?
TrufLL (Task-LM)	what shape is the big thing that is to the right of the big matte thing ?
TrufLL (Ext-LM)	what is the shape of the big object that is behind the big yellow thing and is the same color ?



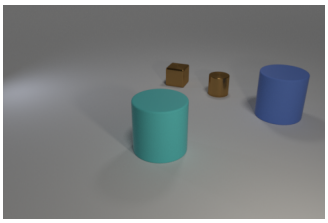
Human	what number of other objects are the same size as the red rubber thing ? **A :3**
pretrain	how many other things are there of the same size as the purple rubber thing ?
pretrain + RL	what number of objects are either large green metallic objects or tiny objects that are behind the tiny purple metalli
scratch	many tiny cubes things things things things things both things as things that
scratch+KL-task	what number of other objects are there of the same material as the tiny cyan thing ?
scratch+KL-ext	are there any blue objects ?
TrufLL (Task-LM)	what number of objects are either big objects in front of the small yellow object or big matte objects ?
TrufLL (Ext-LM)	how many objects in front of the big object ?



Human	what number of other things are there of the same material as the large green object ? **A :3**
pretrain	how many other things are there of the same size as the purple rubber cylinder ?
pretrain + RL	what number of objects are either tiny cyan things or big cyan things ?
scratch	many tiny cubes things things things things things both things as things that
scratch+KL-task	what number of other objects are the same shape as the small yellow object ?
scratch+KL-ext	how many things does that large thing have to its behind ?
TrufLL (Task-LM)	what number of other things are there of the same size as the green cylinder ?
TrufLL (Ext-LM)	how many objects in front of the in the cylinder ?

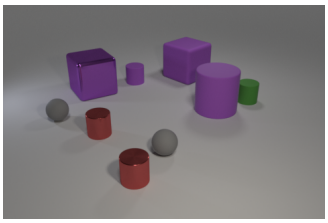


Human	what number of other things are there of the same shape as the small purple metallic thing ? **A :1**
pretrain	what number of other objects are the same color as the tiny rubber cylinder ?
pretrain + RL	what number of purple objects are either small matte objects or big matte blocks ?
scratch	many gray in big purple purple purple many or many gray matte matte
scratch+KL-task	what number of other things are the same color as the large rubber cylinder ?
scratch+KL-ext	how many other things in the are of same color as the large cylinder ?
TrufLL (Task-LM)	how many tiny things have the same color as the large rubber thing ?
TrufLL (Ext-LM)	how many other things in the are of the same color as that large thing ?

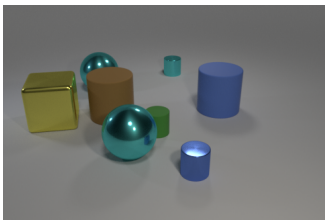


Human	what shape is the big matte object that is on the right side ofthe big cyan matte object ? **A :cylinder**
pretrain	the cyan matte thing that is the same size as the brown object is what shape ?
pretrain + RL	what shape is the cyan matte object that is behind the cylinder ?
scratch	many yellow big either either that that that more that metal ?
scratch+KL-task	what number of other things are the same shape as the small gray thing ?
scratch+KL-ext	what number of blocks are in the things in the ?
TrufLL (Task-LM)	how many tiny things have the same color as the large rubber thing ?
TrufLL (Ext-LM)	what is the shape of that large thing ?

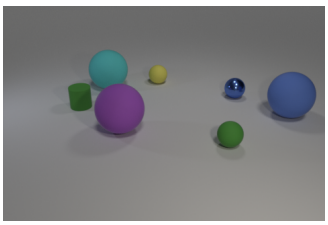
FIGURE A.3 – Samples on CLEVR.



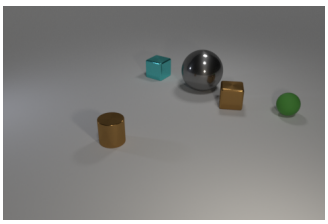
Human	what is the size of the other rubber cylinder that is the same color as the big cylinder ? **A :small**
pretrain	there is a purple object that is the same size as the purple rubber cylinder ; what is its shape ?
pretrain + RL	what size is the gray ball that is right of the purple sphere ?
scratch	that greater tiny as shiny both are a tiny it either ball right
scratch+KL-task	there is a big thing that is the same color as the big matte cylinder ; what is its shape ?
scratch+KL-ext	how material is the yellow ?
TrufLL (Task-LM)	how big is the thing that is to the right of the big matte thing ?
TrufLL (Ext-LM)	what size is the object that is behind the large red thing ?



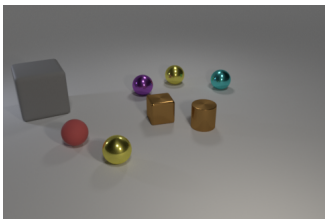
Human	There is a shiny thing that is both right of the small matte thing and behind the large yellow cube ; what size is it ?
pretrain	there is a big thing that is the same color as the small rubber cylinder ; what is its shape
pretrain + RL	there is a brown matte object to the right of the cyan object ; what shape is it ?
scratch	many yellow big either either that that that more that metal ?
scratch+KL-task	what number of other things are the same shape as the small gray thing ?
scratch+KL-ext	what is the material of that block ?
TrufLL (Task-LM)	what shape is the big thing that is to the right of the big cyan thing ?
TrufLL (Ext-LM)	what is the shape of that large thing ?



Human	there is a object that is the same color as the rubber cylinder ;what is its shape ? **A :sphere**
pretrain	there is a small thing that is the same color as the small rubber cylinder ; what is its shape ?
pretrain + RL	what shape is the thing that is the same color as the cylinder ?
scratch	many yellow yellow and cube shape behind cubes shape less small equal shape small equal large large ?
scratch+KL-task	how many other things in the color are of same material as the green shiny object ?
scratch+KL-ext	how many spheres anything ?
TrufLL (Task-LM)	what is the shape of the small cyan thing ?
TrufLL (Ext-LM)	And shape ?



Human	what is the color of the small thing that is the same shape asthe large gray object ? **A :green**
pretrain	there is another rubber object that is the same shape as the small brown object ; what color is it ?
pretrain + RL	what is the color of the tiny rubber thing ?
scratch	many sphere less how an
scratch+KL-task	what number of other objects are the same shape as the tiny blue object ?
scratch+KL-ext	what size is that cylinder ?
TrufLL (Task-LM)	what is the color of the tiny matte thing ?
TrufLL (Ext-LM)	what color is the small thing ?



Human	what number of shiny objects are cyan spheres or tiny balls ? **A :4**
pretrain	how many other things are there of the same size as the brown rubber thing ?
pretrain + RL	how many other things are there of the same size as the cyan rubber thing ?
scratch	many yellow do do do either do either do balls
scratch+KL-task	what number of other things are there of the same shape as the small gray thing ?
scratch+KL-ext	how many other things are of same color as ball ?
TrufLL (Task-LM)	how many other things are the same material as the small cyan cylinder ?
TrufLL (Ext-LM)	how many other things in the material of the small thing that is the same material as green thing ?

FIGURE A.4 – Samples on Clevr.

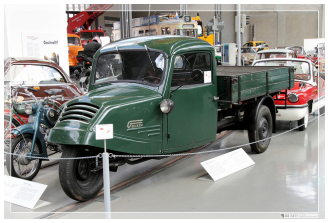
Human	How many trains ? **A :1**
pretrain	How many trains are in the picture ?
pretrain + RL	How many trains are shown ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	How many people are in the picture ?
scratch+KL-ext	The the same way of the same way of the
TrufLL (Task-LM)	How many windows are here ?
TrufLL (Ext-LM)	How many different ways would we take them to reach

Human	What is the man wearing over his shirt ? **A :nothing**
pretrain	What is in front of the person ?
pretrain + RL	What is written on the person's right ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	Is this child old or unloaded ?
scratch+KL-ext	The the first thing of the world's a few
TrufLL (Task-LM)	How many different shapes have been used and used by
TrufLL (Ext-LM)	How many planes can be seen on top ?

Human	Is the fence in the background metal or wood ? **A :metal**
pretrain	What kind of dog is on the dog ?
pretrain + RL	What kind of dog is this ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What is 150 seated over theware ?
scratch+KL-ext	The the same way of the same way to be
TrufLL (Task-LM)	Which item has more aggressive up their legs ?
TrufLL (Ext-LM)	How many different species have been captured and tagged here

Human	What is the title of the red book ? **A :harry potter**
pretrain	What is the cat inside of ?
pretrain + RL	Is the cat inside or outside ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What color is the man's shirt ?
scratch+KL-ext	The way of the world's a lot of the
TrufLL (Task-LM)	Which item appears higher into one and lower ?
TrufLL (Ext-LM)	What was that ?

Human	What color is the fire hydrant ? **A :red**
pretrain	What color is the hydrant ?
pretrain + RL	What color is the hydrant ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What color is the man's bat ?
scratch+KL-ext	The the first thing is a good thing that the
TrufLL (Task-LM)	Which color is this fire ?
TrufLL (Ext-LM)	What color will your feet color look ?



Human	How many wheels does the truck have ? **A :3**
pretrain	How many people are in front of the bus ?
pretrain + RL	How many slices ofists are on the plate ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	Is summer out or cloudy next to Winchester ?
scratch+KL-ext	The the most recent of the most recent years of
TrufLL (Task-LM)	How many pieces are here ?
TrufLL (Ext-LM)	How many different objects have been used ?

FIGURE A.5 – Samples on VQA.

Human	What is on top of the round dome ? **A :flag**
pretrain	What is on the top right mean ?
pretrain + RL	What is on the front of this event ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What zombie is on the mouse ?
scratch+KL-ext	The the first thing is a bit of the first
TrufLL (Task-LM)	Where could one travel park located ?
TrufLL (Ext-LM)	What color will your shoes look ?

Human	Where is the chain link fence ? **A :background**
pretrain	What is he holding ?
pretrain + RL	What is he fire hydrant ?
scratch	mazingAmazingAmazingAmazingAmazing
scratch+KL-task	Who is closest to the paint ?
scratch+KL-ext	The the first thing is a great deal with the
TrufLL (Task-LM)	Which item represents than both ends and lower ?
TrufLL (Ext-LM)	How much food has it given him ?

Human	What activity are these people doing ? **A :surfing**
pretrain	What is the person doing ?
pretrain + RL	What is the person doing ?
scratch	noodles noodles noodles noodles noodles noodles
scratch+KL-task	How many umbrellas are visible ?
scratch+KL-ext	The the first thing is the same way of the
TrufLL (Task-LM)	Which game does he play ?
TrufLL (Ext-LM)	What was that for ?

Human	What color is the umbrella ? **A :black**
pretrain	What color is the cat ?
pretrain + RL	What color is the cat ?
scratch	AmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What color is the man's shirt ?
scratch+KL-ext	The the other way of the past time, and
TrufLL (Task-LM)	Which item doesn't both turn ?
TrufLL (Ext-LM)	What color of clothing did he get ?

Human	How many planes are shown ? **A :1**
pretrain	How many jets are there ?
pretrain + RL	How many jets are there ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	How many skater does Green cents have ?
scratch+KL-ext	The the first thing is the first time, and
TrufLL (Task-LM)	How many surf worthy are here ?
TrufLL (Ext-LM)	How many different ways should one ask if she wants

Human	What is this animal called ? **A :horse**
pretrain	What is the animal on ?
pretrain + RL	What animal is shown on the ground ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What has to make of the pies that, should
scratch+KL-ext	The the next week of the next week, the
TrufLL (Task-LM)	Which item doesn't turn ?
TrufLL (Ext-LM)	What was that ?

FIGURE A.6 – Samples on VQA.

Human	What color spot does the horse have ? **A :white**
pretrain	What color is the animal ?
pretrain + RL	What color is the door ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	What color is the ATM basketball ?
scratch+KL-ext	The the same thing that the same way of the
TrufLL (Task-LM)	Which color is his socks ?
TrufLL (Ext-LM)	What color will your shoes look ?

Human	What color is the girls pants ? **A :blue**
pretrain	What color is the man's blue ?
pretrain + RL	What color are the bird's pants ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-ext	The the first thing is a lot of the same
TrufLL (Task-LM)	Which color is this fire ?
TrufLL (Ext-LM)	What color of clothing did he get ?

Human	What is on the woman's head ? **A :helmet**
pretrain	What is on the girl's head ?
pretrain + RL	What is on the person's head ?
scratch	AmazingAmazingAmazingAmazingAmazingAmazing
scratch+KL-task	Who is behind the horse ?
scratch+KL-ext	The the same thing that the most important to the
TrufLL (Task-LM)	Which item doesn't turn ?
TrufLL (Ext-LM)	What was that ?

FIGURE A.7 – Samples on VQA.

Annexe B

Appendix for Chapter 6

B.1 Details on the training algorithm

B.2 Experiments

B.2.1 Baselines

The LSTM and Transformer with MC Dropout[94] have respectively one and two dropout layers. For the LSTM, this layer is just before the output layer. For the Transformer, the dropout layers are inserted in the network architecture as in [277]. The first dropout layer is after the attention module that computes the output attention vector, and the second one is just before the last layer-normalization layer of the feed-forward neural network that transforms the attention vector. The same dropout rate is kept during training and inference.

The implementation of the Bayesian LSTM relies on the blitz [1] library, following the model from [88]. On the synthetic setting, a hyper-parameter search was performed on prior_{σ_1}, the standard deviation of the first component of the gaussian mixture model that models the prior distribution over the network's weights, and on prior_{π}, the factor to scale this mixture model, as follows :

$$\text{prior}_{\sigma_1} \in \{0.1, 0.135, 0.37, 0.75, 1, 1.5\}, \quad \text{prior}_{\pi} \in \{0.25, 0.5, 0.75, 1\}\,.$$

The best results in terms of predictive performance were obtained for $\text{prior}_{\sigma_1} = 0.1$ and $\text{prior}_{\sigma_1} = 1$: we kept these values for the experiments on the real-world datasets. The other hyper-parameters of the Bayesian LSTM were kept at the default values provided by the blitz library.

1. https ://github.com/piEsposito/blitz-bayesian-deep-learning

B.2.2 Datasets

Synthetic data The synthetic datasets were generated with 1000 samples : we used 800 of them for training and 100 of them for test and validation. A 5-fold cross-validation procedure was performed at training, to estimate the variability in performance that can be attributed to the training algorithm (by opposition with the measurement of the observations variability, computed with the *dist-mse* metric described in Section 4.2 of the main paper).

Real-world time series We use a 0.7 / 0.15 / 0.15 split for training, validation and test for the real-world datasets.

The covid dataset [2] is a univariate time series gathering the daily deaths from the covid-19 disease in 3261 US cities. Cities with less than 100 deaths over the time period considered were discarded from the dataset, leading to 886 samples in the final dataset, with a sequence length equal to 60, corresponding to 2 months of observations.

The air quality dataset [3] gathers hourly responses of a gas multisensor device deployed in an Italian city. It is a multivariate time series with 9 input features : we kept 5 features as target features to be predicted, corresponding to the concentration of 5 chemical gases in the atmosphere. The final dataset has $9,348$ samples, and a sequence length equal to 12, corresponding to a half day of observations.

The weather dataset [4] gathers meteorological data from a German weather station. It is a multivariate time series with 4 input and target features (temperature, air pressure, relative humidity, and air density). The final dataset have $50,000$ samples, and a sequence length equal to 24, corresponding to one day of observations.

The energy dataset [5] gathers 10-min measurements of household appliances energy consumption, coupled with local meteorological data. It is a multivariate time series with 28 input features : we kept 20 target features to be predicted. The final dataset have $19,735$ samples, and a sequence length equal to 12, corresponding to 2 hours of observations.

The stock dataset [6] gathers daily stock prices and volume of General Electric stocks. It is a multivariate time series with 5 input features. The final dataset have $3,020$ samples, and a sequence length equal to 40, corresponding to 2 months of observations (recorded only during business days).

B.2.3 Additional results

Tables B.1 presents additional results on the synthetic datasets for the three settings, and displays the *mse* and *dist-mse* described in section 6.4.2 of the main paper. Table B.2 displays the same metrics for setting I and II, when considering very small variance ($\sigma^2 = 10^{-5}$). In that case, we observe that MC Dropout models and the SMC Transformer tend to exhibit similar *dist-mse*, which slightly overestimate the ground truth (10^{-5}), while the Bayesian LSTM exhibits a higher variability.

2. https ://github.com/CSSEGISandData/COVID-19
3. https ://archive.ics.uci.edu/ml/datasets/air+quality
4. https ://www.bgc-jena.mpg.de/wetter/Weatherstation.pdf
5. https ://archive.ics.uci.edu/ml/datasets/Appliances+energy+prediction
6. https ://www.kaggle.com/szrlee/stock-time-series-20050101-to-20171231 ?select=GE_2006-01-01_to_2018-01-01.csv

TABLE B.1 – Mean Square Error of the mean predictions (mse) and Mean Square Error of the predictive distribution (dist-mse) on the test set versus the ground truth, for setting I, II, and III. Values are computed with a 5-fold cross-validation procedure on each dataset. Standard deviations are displayed in parenthesis when they are larger than 0.01. For the LSTM and Transformer models with MC Dropout, p is the dropout rate. For the Bayesian LSTM, M is the number of Monte Carlo samples to estimate the ELBO loss [88]. We display the results for $M = 3$, as it is the default parameter value provided by the blitz library. For the SMC Transformer, M is the number of particles of the SMC algorithm.

	Model I		**Model II**		**Model III**	
Model	mse	dist-mse	mse	dist-mse	mse	dist-mse
True Model	0.5	0.50(0.03)	0.3	0.35(0.07)	-	1.56
LSTM	**0.50**	N.A	0.32	N.A	0.20 (0.02)	N.A
Transformer	0.52	N.A	0.32	N.A	0.26 (0.02)	N.A
LSTM drop.						
$p = 0.1$	0.48	0.004	0.32	0.003	0.19 (0.05)	1.09 (0.05)
$p = 0.2$	0.53	0.0099	0.34	0.007	0.22 (0.05)	1.26 (0.04)
$p = 0.5$	0.53	0.03	0.33	0.02	0.22 (0.02)	1.23 (0.02)
Transf. drop.						
$p = 0.1$	**0.50**	0.02	0.31	0.03	0.26 (0.03)	1.03 (0.04)
$p = 0.2$	0.50	0.02(0.01)	0.32	0.02	0.29 (0.02)	1.16 (0.04)
$p = 0.5$	0.52(0.01)	0.05(0.02)	0.33(0.02)	0.05(0.02)	0.38 (0.03)	1.07 (0.03)
Bayes. LSTM						
$M = 3$	0.55(0.01)	0.04	0.36(0.01)	0.05	0.42 (0.02)	1.18 (0.05)
$M = 10$	0.53(0.01)	0.03	0.37(0.01)	0.04	0.35 (0.01)	1.11 (0.03)
SMC Transf.						
$M = 10$	0.52	**0.49**	**0.30**	**0.35**	0.35	**1.35** (0.04)
$M = 30$	0.49	0.52	0.34	**0.35**	0.35	1.32 (0.02)

Table B.3 and Table B.4 present the additional results when doing respectively *unistep forecasting* and *multi-step forecasting*, and display the mean square error over the test set (respectively named *mse* for the unistep case and *mse-m* for the multi-step case), and the predictive interval metrics (PICP and MPIW) described in section section 6.4.3 of the main paper. For the multivariate time series, the PICP and MPIW are averaged over all the target features.

TABLE B.2 – Mean Square Error of the mean predictions (mse) and Mean Square Error of the predictive distribution (dist-mse) on the test set versus the ground truth, **for settings I and II, with very small variance ($\sigma^2 = 10^{-5}$)**. Values are computed with a 5-fold cross-validation procedure on each dataset. Standard deviations are displayed in parenthesis when they are larger than 0.01. For the LSTM and Transformer models with MC Dropout, p is the dropout rate. For the Bayesian LSTM, M is the number of Monte Carlo samples to estimate the ELBO loss [88]. For the SMC Transformer, M is the number of particles of the SMC algorithm.

	Setting I		Setting II	
	mse	dist-mse	mse	dist-mse
True Model	10^{-5}	10^{-5}	10^{-5}	
LSTM	$1.6 * 10^{-5}$	N.A	**0.003**	N.A
Transformer	$8.1 * 10^{-5}$	N.A	**0.003**	N.A
LSTM drop.				
$p = 0.1$	$3.2 * 10^{-4}$	$3.4 * 10^{-4}$	**0.003**	0.004
$p = 0.2$	$7.6 * 10^{-4}$	$7.2 * 10^{-4}$	**0.003**	**0.003**
$p = 0.5$	$2.1 * 10^{-3}$	$3.5 * 10^{-5}$	0.006	0.006
Transf. drop.				
$p = 0.1$	$3.1 * 10^{-4}$	$1.0 * 10^{-3}$	0.004	0.004
$p = 0.2$	$6.9 * 10^{-4}$	$2.0 * 10^{-3}$	0.004	0.005
$p = 0.5$	$3.0 * 10^{-3}$	$3.2 * 10^{-3}$	0.005	0.008
Bayes. LSTM				
$M = 3$	$1.3 * 10^{-2}$	$1.1 * 10^{-2}$	0.017	0.018
$M = 10$	$7.9 * 10^{-3}$	$9.1 * 10^{-3}$	0.010	0.011
SMC Transf.				
$M = 10$	$2.2 * 10^{-4}$	$7.6 * 10^{-4}$	**0.003**	0.007
$M = 30$	$2.9 * 10^{-4}$	$9.5 * 10^{-4}$	**0.003**	0.008

TABLE B.3 – Mean Square Error (test loss), PICP and MPIW for unistep forecast. The underlined values correspond to the best performances. MPIW are only effective when the PICP is valid (above 0.95 - green). MPIW associated with poor PICP (below 0.90 - red) are grayed. For the transformer architectures, L is the number of layers and H is the number of heads.

	Covid		Air quality		Weather		Energy		Stock	
	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw	mse	picp | mpiw
LSTM	0.117	- | -	0.120	- | -	0.080	- | -	0.039	- | -	0.055	- | -
LSTM drop.										
$p = 0.1$	0.150	0.77 | 0.27	0.139	0.64 | 0.40	0.097	0.61 | 1.11	0.07	0.99 | 0.49	0.065	0.93 | 0.28
$p = 0.2$	0.159	0.79 | 0.38	0.152	0.72 | 0.54	0.109	0.68 | 1.37	0.103	0.99 | 0.45	0.074	0.95 | 0.38
$p = 0.5$	0.155	0.89 | 0.63	0.211	0.86 | 0.87	0.165	0.79 | 0.92	0.218	0.96 | 1.24	0.112	0.97 | 0.68
Transf.	0.116	- | -	0.132	- | -	0.110	- | -	0.042	- | -	0.068	- | -
Transf drop.										
$p = 0.1, L = 1, H = 1$	0.121	0.96 | 0.37	0.141	0.77 | 0.47	0.130	0.65 | 0.37	0.046	0.96 | 0.45	0.076	0.93 | 0.34
$p = 0.2, L = 1, H = 1$	0.129	0.94 | 0.46	0.159	0.89 | 0.58	0.135	0.74 | 0.47	0.053	0.98 | 0.64	0.082	0.95 | 0.41
$p = 0.1, L = 1, H = 1$	0.208	0.97 | 0.76	0.196	0.96 | 0.85	0.209	0.89 | 0.65	0.09	0.97 | 1.00	0.106	0.98 | 0.63
Bayesian LSTM	0.144	0.25 | 0.12	0.192	0.77 | 0.51	0.113	0.44 | 0.24	0.121	0.91 | 0.76	0.086	0.85 | 0.22
SMC Transf.										
$L = 1, H = 1$	0.128	0.997 | 0.70	0.148	0.97 | 1.54	0.181	0.99 | 1.66	0.043	0.99 | 0.82	0.071	0.99 | 1.08
$L = 2, H = 4$	0.153	0.99 | 0.62	0.123	0.96 | 1.36	0.126	0.96 | 1.45	0.041	0.99 | 0.79	0.066	0.99 | 0.98

TABLE B.4 – Multi-step Mean Square Error (mse-m), and multistep forecast PICP and MPIW. The underlined values correspond to the best performances. MPIW are only effective when the PICP is valid (above 0.95 - green). MPIW associated with poor PICP (below 0.90 - red) are grayed. For the transformer architectures, L is the number of layers and H is the number of heads.

	Covid		Air quality		Weather		Energy		Stock	
	mse-m	picp | mpiw	mse-m	picp | mpiw	mse-m	picp | mpiw	mse-m	picp | mpiw	mse-m	picp | mpiw
LSTM drop.										
$p = 0.1$	0.852	0.67 | 0.61	0.339	0.54 | 0.73	0.418	0.61 | 1.11	0.116	0.96 | 1.12	0.117	0.85 | 0.74
$p = 0.2$	0.914	0.73 | 0.83	0.463	0.61 | 0.97	0.459	0.68 | 1.37	0.167	0.96 | 1.34	0.140	0.91 | 1.05
$p = 0.5$	0.697	0.80 | 1.64	0.501	0.70 | 1.31	0.559	0.75 | 1.68	0.349	0.88 | 1.57	0.277	0.87 | 1.42
Transf. drop.										
$p = 0.1$, $L = 1$, $H = 1$	0.777	0.74 | 0.74	0.623	0.69 | 1.44	0.426	0.42 | 0.67	0.108	0.89 | 0.60	0.167	0.71 | 0.54
$p = 0.2$, $L = 1$, $H = 1$	0.722	0.84 | 1.51	0.642	0.70 | 0.16	0.414	0.50 | 0.79	0.119	0.93 | 0.77	0.172	0.72 | 0.51
$p = 0.5$, $L = 1$, $H = 1$	0.708	0.84 | 1.52	0.827	0.77 | 1.97	0.485	0.61 | 1.11	0.174	0.94 | 1.21	0.185	0.83 | 0.77
Bayesian LSTM	0.490	0.15 | 0.23	0.372	0.49 | 0.72	0.461	0.36 | 0.54	0.155	0.93 | 1.08	0.276	0.34 | 0.45
SMC Transf.										
$L = 1$, $H = 1$	0.529	0.91 | 1.85	0.548	0.97 | 3.17	0.510	0.92 | 2.93	0.113	0.97 | 1.33	0.206	0.98 | 1.80
$L = 2$, $H = 4$	0.507	0.89 | 1.20	0.652	0.90 | 2.54	0.550	0.93 | 2.82	0.115	0.98 | 1.30	0.225	0.96 | 1.34

Annexe C

Appendix of Chapter 7

C.1 Application to partially observed SDE

Let $(X_t)_{t\geq 0}$ be defined as a weak solution to the following Stochastic Differential Equation (SDE) in $\mathbb{R}^d$:

$$X_0 = x_0 \quad \text{and} \quad \mathrm{d}X_t = \alpha_\theta(X_t)\mathrm{d}t + \mathrm{d}W_t \ , \tag{C.1}$$

where $(W_t)_{t\geqslant 0}$ is a standard Brownian motion, $\alpha_\theta : \mathbb{R}^d \to \mathbb{R}^d$ is the drift function. The inference procedure presented in this work is applied in the case where the solution to (C.1) is supposed to be partially observed at times $t_0 = 0, \ldots, t_n$, for a given $n \geqslant 1$, through an observation process $(Y_k)_{0\leqslant k\leqslant n}$ taking values in $\mathbb{R}^m$. For all $0 \leqslant k \leqslant n$, the distribution of Y_k given $(X_t)_{t\geqslant 0}$ depends on $X_k = X_{t_k}$ only and has density $g_{k;\theta}$ with respect to ν. The distribution of X_0 has density χ with respect to μ and for all $0 \leqslant k \leqslant n-1$, the conditional distribution of X_{k+1} given $(X_t)_{0\leqslant t\leqslant k}$ has density $q_{k+1;\theta}(X_k, \cdot)$ with respect to μ. The algorithm described in this document strongly relies on assumption H1. In the context of SDEs, when $g_{k+1;\theta}$ is available explicitly, this boils down to finding an unbiased estimate $\widehat{q}_{k+1;\theta}\langle\zeta\rangle(x_k, x_{k+1})$ of $q_{k+1;\theta}(x_k, x_{k+1})$ and defining

$$\ell_{k;\theta}\langle\zeta\rangle(x_k, x_{k+1}) = \widehat{q}_{k+1;\theta}\langle\zeta\rangle(x_k, x_{k+1}) g_{k+1;\theta}(x_{k+1}, Y_{k+1}) \ .$$

C.1.1 General Poisson Estimators

In [211] and [102], General Poisson Estimators (GPEs) are used to obtain an unbiased estimate of the transition density. However, designing such estimators requires several strong assumptions [20].

1. The drift of the unit diffusion can be expressed as the gradient of a potential function, i.e., there exists a twice differentiable function $A_\theta : \mathbb{R}^d \to \mathbb{R}$ such $\alpha_\theta = \nabla_x A_\theta$.
2. The function $x \mapsto (\|\alpha_\theta(x)\|^2 + \Delta A_\theta(x))/2$ (where Δ denotes the Laplacian) is lower bounded.

The fact that the target SDE has a unit diffusion term is used to define a proposal distribution absolutely continuous with respect to the target which is easy to sample from. The first assumption is necessary to obtain a tractable Radon-Nikodym derivative between the proposal and the target distributions using the Girsanov transformation. While these assumptions can be proved under mild assumptions for scalar diffusions, much stronger conditions are required in the multidimensional case [3]. Let $\omega = (\omega_s)_{0\leq s\leq t}$ be the realization of a Brownian Bridge starting at x at time 0 and ending at y at time Δ_k, where $\Delta_k = t_{k+1} - t_k$. The distribution of ω is denoted by $\mathbb{W}_x^{\Delta_k,y}$. Writing, $\psi_\theta :\ x \mapsto \psi_\theta(x) = (\|\alpha_\theta(x)\|^2 + \Delta A_\theta(x))/2$, by Girsanov theorem, for all $x, y \in \mathbb{R}^d \times \mathbb{R}^d$

$$q_{k+1;\theta}(x,y) = \phi_{\Delta_k}(x-y)\exp\left(A_\theta(y) - A_\theta(x)\right)\mathbb{E}_{\mathbb{W}_x^{\Delta_k,y}}\left[\exp\left(-\int_0^{\Delta_k}\psi_\theta(\omega_s)\mathrm{d}s\right)\right] , \tag{C.2}$$

where for all $a > 0$, ϕ_a is the probability density function of a centered Gaussian random variable with variance a. The transition density then cannot be computed as it involves an integration over the whole path between x and y. To perform the algorithm proposed in this document, we therefore have to design a positive an unbiased estimator of $q_{k+1;\theta}(x,y)$.

Proposition 1 (Unbiased GPE estimator $\widehat{q}_{k+1;\theta}\langle\zeta\rangle(x,y)$)**.** *Let $\omega = (\omega_s)_{0\leq s\leq\Delta_k}$ be the realization of the Brownian Bridge starting at x at time 0 and ending at y at time Δ_k. Assume that there exist random variables $\underline{m}_\theta$ and $\overline{m}_\theta$ such that for all $0 \leqslant s \leqslant \Delta_k$, $\underline{m}_\theta \leqslant \psi_\theta(\omega_s) \leqslant \overline{m}_\theta$. Let κ be a Poisson random variable with parameter $(\overline{m}_\theta - \underline{m}_\theta)\Delta_k$. Let $(U_j)_{1\leqslant j\leqslant\kappa}$ be independent uniform random variables on $(0,\Delta_k)$ and $\zeta = (\kappa, \omega, U_1, \ldots, U_\kappa)$. A positive unbiased estimator of $q_{k;\theta}(x,y)$ is*

$$\widehat{q}_{k+1;\theta}\langle\zeta\rangle(x,y) = \phi_{\Delta_k}(x-y)\exp\left(A_\theta(y) - A_\theta(x) - \underline{m}_\theta\Delta_k\right)\prod_{j=1}^{\kappa}\frac{\overline{m}_\theta - \psi_\theta(\omega_{U_j})}{\overline{m}_\theta - \underline{m}_\theta} .$$

Proof. The proof is adapted from [79] and given here for completeness. Let $\omega = (\omega_s)_{0\leq s\leq\Delta_k}$ be the realization of the Brownian Bridge. Note first that, if κ is a Poisson random variable with parameter $(\overline{m}_\theta - \underline{m}_\theta)\Delta_k$,

$$\begin{aligned}
\exp\left(-\int_0^{\Delta_k}\psi_\theta(\omega_s)\mathrm{d}s\right) &= \mathrm{e}^{-\overline{m}_\theta\Delta_k}\sum_{j\geq 0}\frac{1}{j!}\left((\overline{m}_\theta - \underline{m}_\theta)\Delta_k\int_0^{\Delta_k}\frac{\overline{m}_\theta - \psi_\theta(\omega_s)}{(\overline{m}_\theta - \underline{m}_\theta)\Delta_k}\mathrm{d}s\right)^j , \\
&= \mathrm{e}^{((\overline{m}_\theta - \underline{m}_\theta) - \overline{m}_\theta)\Delta_k}\mathbb{E}\left[\left(\int_0^{\Delta_k}\frac{\overline{m}_\theta - \psi_\theta(\omega_s)}{(\overline{m}_\theta - \underline{m}_\theta)\Delta_k}\mathrm{d}s\right)^\kappa\middle|\omega\right] , \\
&= \mathrm{e}^{-\underline{m}_\theta\Delta_k}\mathbb{E}\left[\left(\int_0^{\Delta_k}\frac{\overline{m}_\theta - \psi_\theta(\omega_s)}{(\overline{m}_\theta - \underline{m}_\theta)\Delta_k}\mathrm{d}s\right)^\kappa\middle|\omega\right] .
\end{aligned}$$

Therefore, if $\kappa \sim \mathcal{P}((\overline{m}_\theta - \underline{m}_\theta)\Delta_k)$, and $(U_j)_{1\leqslant j\leqslant\kappa}$ are independent uniform random variables on $(0,\Delta_k)$ and $\zeta =$

$(\kappa, \omega, U_1, \ldots, U_\kappa)$, by (C.2),

$$\widehat{q}_{k+1;\theta}\langle\zeta\rangle(x,y) = \phi_{\Delta_k}(x-y)\exp\left(A_\theta(y) - A_\theta(x) - \underline{m}_\theta \Delta_k\right) \prod_{j=1}^{\kappa} \frac{\overline{m}_\theta - \psi_\theta(\omega_{U_j})}{\overline{m}_\theta - \underline{m}_\theta}$$

is an unbiased estimator of $q_{k+1;\theta}(x,y)$. □

Proposition 2 (Unbiased GPE estimator of $\nabla_\theta \log q_{k+1;\theta}(x,y)$)**.** *Assume that there exist random variables $\underline{m}_\theta$ and $\overline{m}_\theta$ such that for all $0 \leqslant s \leqslant \Delta_k$, $\underline{m}_\theta \leqslant \psi_\theta(\omega_s) \leqslant \overline{m}_\theta$. Let $\mathbb{S}_{\theta,x}^{\Delta_k,y}$ be the diffusion bridge associated with the SDE* (C.1) *starting at x and ending at y at time Δ_k. Then, an unbiased estimator of $\nabla_\theta \log q_{k+1;\theta}(x,y)$ is given by*

$$\mathsf{I}_{k+1;\theta}(x,y,\mathsf{s}_U^{\theta,x,y,\Delta_k}) = (\nabla_\theta A_\theta(y) - \nabla_\theta A_\theta(x) - \nabla_\theta \underline{m}_\theta \Delta_k) - \Delta_k \nabla_\theta \varphi_\theta(\mathsf{s}_U^{\theta,x,y,\Delta_k}) \,,$$

where U is uniform on $(0,1)$ and independent of $\mathsf{s}^{\theta,x,y,\Delta_k} \sim \mathbb{S}_{\theta,x}^{\Delta_k,y}$.

Proof. Define $\varphi_\theta : \; x \mapsto \psi_\theta(x) - \underline{m}_\theta$. By (C.2),

$$\nabla_\theta \log q_{k+1;\theta}(x,y) = \nabla_\theta A_\theta(y) - \nabla_\theta A_\theta(x) - \nabla_\theta \underline{m}_\theta \Delta_k - \frac{\mathbb{E}_{\mathbb{W}_x^{\Delta_k,y}}\left[\left(\int_0^{\Delta_k} \nabla_\theta \varphi_\theta(\omega_s)\mathrm{d}s\right)\exp\left(-\int_0^{\Delta_k}\varphi_\theta(\omega_s)\mathrm{d}s\right)\right]}{\mathbb{E}_{\mathbb{W}_x^{\Delta_k,y}}\left[\exp\left(-\int_0^{\Delta_k}\varphi_\theta(\omega_s)\mathrm{d}s\right)\right]} \,.$$

On the other hand, the diffusion bridge $\mathbb{S}_{\theta,x}^{\Delta_k,y}$ associated with the SDE (C.1) is absolutely continuous with respect to $\mathbb{W}_x^{\Delta_k,y}$ with Radon-Nikodym derivative given by

$$\begin{aligned}\frac{\mathrm{d}\mathbb{S}_{\theta,x}^{\Delta_k,y}}{\mathrm{d}\mathbb{W}_x^{\Delta_k,y}}(\omega) &= \left[q_{k+1;\theta}(x,y)\right]^{-1} \phi_{\Delta_k}(x-y)\exp\left(A_\theta(y) - A_\theta(x) - \underline{m}_\theta\Delta_k - \int_0^{\Delta_k}\varphi_\theta(\omega_s)\mathrm{d}s\right) \,, \\ &= \mathbb{E}_{\mathbb{W}_x^{\Delta_k,y}}\left[\exp\left(-\int_0^{\Delta_k}\varphi_\theta(\omega_s)\mathrm{d}s\right)\right]^{-1} \exp\left(-\int_0^{\Delta_k}\varphi_\theta(\omega_s)\mathrm{d}s\right) \,.\end{aligned}$$

This yields

$$\nabla_\theta \log q_{k+1;\theta}(x,y) = (\nabla_\theta A_\theta(y) - \nabla_\theta A_\theta(x) - \nabla_\theta \underline{m}_\theta \Delta_k) - \mathbb{E}_{\mathbb{S}_{\theta,x}^{\Delta_k,y}}\left[\int_0^{\Delta_k} \nabla_\theta \varphi_\theta(\omega_s)\mathrm{d}s\right]$$

and an unbiased estimator of $\nabla_\theta \log q_{k+1;\theta}(x,y)$ is given by

$$\mathsf{I}_{k+1;\theta}(x,y,\mathsf{s}_U^{\theta,x,y,\Delta_k}) = (\nabla_\theta A_\theta(y) - \nabla_\theta A_\theta(x) - \nabla_\theta \underline{m}_\theta \Delta_k) - \Delta_k \nabla_\theta \varphi_\theta(\mathsf{s}_U^{\theta,x,y,\Delta_k}) \,,$$

where U is uniform on $(0,1)$ and independent of $\mathsf{s}^{\theta,x,y,\Delta_k} \sim \mathbb{S}_{\theta,x}^{\Delta_k,y}$. In the context of GPE, $\mathsf{s}^{\theta,x,y,\Delta_k}$ can be simulated exactly using exact algorithms for diffusion processes proposed in [20]. □

C.1.2 Parametrix estimators

More recently, [6] and [83] proposed an algorithm which can be used under weaker assumptions. This parametrix algorithm draws weighted skeletons using an importance sampling mechanism for diffusion processes. In this case, the sampled paths are not distributed as the target process but the weighted samples produce unbiased estimates of expectations of functionals of this process. To obtain an unbiased estimator $\widehat{q}_{k+1}\langle\zeta\rangle(x,y)$, the parametrix algorithm draws weighted skeletons at random times $s_0 = 0 < s_1 < \cdots < s_j$, denoted by $\left\{(x_{s_j}, \mathsf{w}_{s_j})\right\}_{j\geqslant 0}$, where $x_0 = x$ and $\mathsf{w}_0 = 1$. The update times $(s_j)_{j\geqslant 0}$ are instances of an inhomogeneous Poisson process of intensity $\lambda(t)$. Let $(x_{s_j}, \mathsf{w}_{s_j})$ be the last weighted sample and s_{j+1} be the next update time of the trajectory. While $s_{j+1} < \Delta t_k$, the new state is sampled using a simple Euler scheme, namely :

$$x_{s_{j+1}} := x_{s_j} + \Delta s_j \alpha_\theta(x_{s_j}) + (\Delta s_j)^{1/2}\sigma_\theta(x_{s_j})\varepsilon_{j+1} \ ,$$

where $\Delta s_j := s_{j+1} - s_j$, $\Delta t_k = t_{k+1} - t_k$ and $\varepsilon_{j+1} \sim \mathcal{N}_d(0, I_d)$. The proposal density associated with this procedure is denoted by $m_{j;\theta}\left(x_{s_j}, \cdot, \Delta\tau_j\right)$. Let $\mathcal{K}^\theta$ (resp. $\mathcal{K}^{j,\theta}_{\mathrm{prop}}$) denote the Kolmogorov forward operator of the diffusion (resp. the Kolmogorov forward operator of the proposal distribution $m_{j;\theta}\left(x_{s_j}, \cdot, \Delta s_j\right)$). The forward operators write, for any function $h : \mathbb{R}^d \to \mathbb{R}$,

$$\mathcal{K}^\theta h\left(y\right) := -\sum_{i=1}^{d} \frac{\partial}{\partial y_i}\left\{\alpha_{\theta,i}(y)h\left(y\right)\right\} + \sum_{i,\ell=1}^{d} \frac{1}{2}\frac{\partial^2}{\partial y_i \partial y_\ell}\left\{\gamma_{\theta,i,\ell}(y)h\left(y\right)\right\} \ ,$$

where $\gamma_\theta = \sigma_\theta\sigma_\theta^T$. Then, following [83], the weight is updated by

$$\mathsf{w}_{s_{j+1}} := \mathsf{w}_{s_j}\rho_j^\lambda\left(x_{s_j}, x_{s_{j+1}}, \Delta s_j\right) \ ,$$

where

$$\rho_j^\lambda\left(x,y,u\right) := 1 + \frac{\left(\mathcal{K} - \mathcal{K}^{j,\theta}_{\mathrm{prop}}\right) m_{j;\theta}\left(x,z,u\right)_{|z=y}}{\lambda(u) m_{j;\theta}\left(x,y,u\right)} \ . \tag{C.3}$$

It is worth noting that (C.3) can be computed using only first derivatives of α_θ and second derivatives of σ_θ. If N_k is the number of Poisson events between 0 and Δt_k, the parametrix unbiased estimate is then given by

$$\widehat{q}_{k+1}\langle\zeta_k\rangle(x,y) = \mathsf{w}_{s_{N_k}} m_{k;\theta}\left(x_{s_{N_k}}, y, t_{k+1} - s_{N_k}\right) \ ,$$

where ζ_k stands for all the randomness required to produce the parametrix estimator (Poisson process and Gaussian random variables).

The stability of this estimator is studied in [83] which provides L_p controls for the weight $\mathsf{w}_{s_{N_k}}$. The parametrix algorithm mentioned above is a highly flexible procedure to obtain such an unbiased estimate for a much broader

class of diffusions than Poisson based estimations which require strong assumptions. However, as the update (C.3) involves the difference of two Kolmogorov operators, the parametrix estimator of the transition density may be negative, and has no reason to satisfy (7.12).

Bibliographie

[1] S. Abney. Statistical methods and linguistics. *The balancing act : Combining symbolic and statistical approaches to language*, pages 1–26, 1996.

[2] Y. Achbany, F. Fouss, L. Yen, A. Pirotte, and M. Saerens. Tuning continual exploration in reinforcement learning : An optimality property of the boltzmann strategy. *Neurocomputing*, 71(13-15) :2507–2520, 2008.

[3] Y. Aït-Shalia. Closed-form likelihood expansions for multivariate diffusions. *Annals of Statistics*, 36(2) :906–937, 2008.

[4] V. B. Alex Kendall and R. Cipolla. Bayesian segnet : Model uncertainty in deep convolutional encoder-decoder architectures for scene understanding. In *Proceedings of the British Machine Vision Conference (BMVC)*, 2017.

[5] P. Ammanabrolu and M. O. Riedl. Playing text-adventure games with graph-based deep reinforcement learning. In *Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2018.

[6] P. Andersson and A. Kohatsu-Higa. Unbiased simulation of stochastic differential equations using parametrix expansions. *Bernoulli*, 23(3) :2028–2057, 2017.

[7] C. Andrieu and G. O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2) :697 – 725, 2009. doi : 10.1214/07-AOS574. URL https://doi.org/10.1214/07-AOS574.

[8] A. Ashukha, A. Lyzhov, D. Molchanov, and D. Vetrov. Pitfalls of in-domain uncertainty estimation and ensembling in deep learning. In *International Conference on Learning Representations (ICLR)*, 2020.

[9] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv :1607.06450*, 2016.

[10] L. J. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *CoRR*, abs/1607.06450, 2016. URL http://arxiv.org/abs/1607.06450.

[11] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2015.

[12] D. Bahdanau, P. Brakel, K. Xu, A. Goyal, R. Lowe, J. Pineau, A. Courville, and Y. Bengio. An actor-critic algorithm for sequence prediction. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[13] D. Bahdanau, H. de Vries, T. J. O'Donnell, S. Murty, P. Beaudoin, Y. Bengio, and A. Courville. Closure : Assessing systematic generalization of clevr models. *Visually Grounded Interaction and Language (ViGIL)*, 2019.

[14] H. Bahuleyan, L. Mou, O. Vechtomova, and P. Poupart. Variational attention for sequence-to-sequence models. *arXiv preprint arXiv :1712.08207*, 2017.

[15] L. Baird. Residual algorithms : Reinforcement learning with function approximation. In *Machine Learning Proceedings 1995*, pages 30–37. Elsevier, 1995.

[16] S. Banerjee and A. Lavie. Meteor : An automatic metric for mt evaluation with improved correlation with human judgments. In *Proc. of ACL Workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*, 2005.

[17] L. W. Barsalou, A. Santos, W. K. Simmons, and C. D. Wilson. Language and simulation in conceptual processing. *Symbols, embodiment, and meaning*, pages 245–283, 2008.

[18] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Proc. of Neural Information Processing Systems (NIPS)*, 2015.

[19] D. P. Bertsekas and J. N. Tsitsiklis. Neuro-dynamic programming : an overview. In *Proceedings of 1995 34th IEEE conference on decision and control*, volume 1, pages 560–564. IEEE, 1995.

[20] A. Beskos, O. Papaspiliopoulos, and G. O. Roberts. Retrospective exact simulation of diffusion sample paths with applications. *Bernoulli*, 12(6) :1077–1098, 2006.

[21] A. Beskos, O. Papaspiliopoulos, G. O. Roberts, and P. Fearnhead. Exact and computationally efficient likelihood-based estimation for discretely observed diffusion processes (with discussion). *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 68(3) :333–382, 2006.

[22] Y. Bisk, A. Holtzman, J. Thomason, J. Andreas, Y. Bengio, J. Chai, M. Lapata, A. Lazaridou, J. May, A. Nisnevich, N. Pinto, and J. Turian. Experience grounds language, 2020.

[23] M. Blank and D. Lebling. Zork i : The great underground empire. Infocom, 1980.

[24] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference : A review for statisticians. *CoRR*, abs/1601.00670, 2016. URL http://arxiv.org/abs/1601.00670.

[25] D. M. Blei, A. Kucukelbir, and J. D. McAuliffe. Variational inference : A review for statisticians. *Journal of the American statistical Association*, 112(518) :859–877, 2017.

[26] S. L. Blodgett, L. Green, and B. O'Connor. Demographic dialectal variation in social media : A case study of african-american english. *arXiv preprint arXiv :1608.08868*, 2016.

[27] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra. Weight uncertainty in neural networks. *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[28] R. Bommasani, D. A. Hudson, E. Adeli, R. Altman, S. Arora, S. von Arx, M. S. Bernstein, J. Bohg, A. Bosselut, E. Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv :2108.07258*, 2021.

[29] S. Bond-Taylor, A. Leach, Y. Long, and C. G. Willcocks. Deep generative modelling : A comparative review of vaes, gans, normalizing flows, energy-based and autoregressive models. *arXiv preprint arXiv :2103.04922*, 2021.

[30] S. R. Bowman, L. Vilnis, O. Vinyals, A. Dai, R. Jozefowicz, and S. Bengio. Generating sentences from a continuous space. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 10–21, Berlin, Germany, Aug. 2016. Association for Computational Linguistics. doi : 10.18653/v1/K16-1002. URL https://aclanthology.org/K16-1002.

[31] M. Briers, A. Doucet, and S. Maskell. Smoothing algorithms for state–space models. *Annals of the Institute of Statistical Mathematics*, 62(1) :61, 2010.

[32] P. F. Brown, S. A. Della Pietra, V. J. Della Pietra, and R. L. Mercer. The mathematics of statistical machine translation : Parameter estimation. *Computational Linguistics*, 19(2) :263–311, 1993. URL https://aclanthology.org/J93-2003.

[33] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Proc. of Neural Information Processing Systems (NeurIPS)*, 2020.

[34] J. Bruner. Child's talk : Learning to use language. *Child Language Teaching and Therapy*, 1(1) :111–114, 1985.

[35] M. Caccia, L. Caccia, W. Fedus, H. Larochelle, J. Pineau, and L. Charlin. Language gans falling short. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[36] C. Campbell, N. Cristianini, and A. J. Smola. Query learning with large margin classifiers. In *Proceedings of International Conference on Machine Learning (ICML)*, 2000.

[37] L. M. Candanedo, V. Feldheim, and D. Deramaix. A methodology based on hidden markov models for occupancy detection and a case study in a low energy residential building. *Energy and Buildings*, 148 :327–341, 2017.

[38] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models*. Springer, 2005.

[39] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models*. Springer, 2005.

[40] O. Cappé, S. J. Godsill, and E. Moulines. An overview of existing methods and recent advances in sequential monte carlo. *Proceedings of the IEEE*, 95(5) :899–924, 2007.

[41] J. Carpenter, P. Clifford, and P. Fearnhead. An improved particle filter for non-linear problems. *IEE Proceedings Radar Sonar and Navigation*, 146(1) :2–7, Feb. 1999. ISSN 1350-2395. doi : 10.1049/ip-rsn:19990255.

[42] R. Chaabouni, E. Kharitonov, D. Bouchacourt, E. Dupoux, and M. Baroni. Compositionality and generalization in emergent languages. *arXiv preprint arXiv :2004.09124*, 2020.

[43] Y. Chandak, G. Theocharous, J. Kostas, S. Jordan, and P. S. Thomas. Learning action representations for reinforcement learning. In *Proc. of International Conference on Machine Learning (ICML)*, 2019.

[44] S. Chaudhari, V. Mithal, G. Polatkan, and R. Ramanath. An attentive survey of attention models. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 12(5) :1–32, 2021.

[45] S. Chen, Y. Hou, Y. Cui, W. Che, T. Liu, and X. Yu. Recall and learn : Fine-tuning deep pretrained language models with less forgetting. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[46] K. Cho, B. van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. pages 1724–1734, Oct. 2014. doi : 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

[47] L. Choshen, L. Fox, Z. Aizenbud, and O. Abend. On the weaknesses of reinforcement learning for neural machine translation. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2020.

[48] G. G. Chowdhury. Natural language processing. *Annual review of information science and technology*, 37 (1) :51–89, 2003.

[49] J. Chung, K. Kastner, L. Dinh, K. Goel, A. C. Courville, and Y. Bengio. A recurrent latent variable model for sequential data. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2015.

[50] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov. Unsupervised cross-lingual representation learning at scale. *arXiv preprint arXiv :1911.02116*, 2019.

[51] A. Das, S. Kottur, K. Gupta, A. Singh, D. Yadav, J. M. Moura, D. Parikh, and D. Batra. Visual dialog. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[52] A. Das, S. Kottur, J. M. Moura, S. Lee, and D. Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In *Proc. of International Conference on Computer Vision (ICCV)*, 2017.

[53] S. Dathathri, A. Madotto, J. Lan, J. Hung, E. Frank, P. Molino, J. Yosinski, and R. Liu. Plug and play language models : A simple approach to controlled text generation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=H1edEyBKDS.

[54] C. d. M. d'Autume, M. Rosca, J. Rae, and S. Mohamed. Training language gans from scratch. In *Proc. of Neural Information Processing Systems (NeurIPS)*, 2019.

[55] H. De Vries, F. Strub, S. Chandar, O. Pietquin, H. Larochelle, and A. Courville. Guesswhat ?! visual object discovery through multi-modal dialogue. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[56] H. de Vries, K. Shuster, D. Batra, D. Parikh, J. Weston, and D. Kiela. Talk the walk : Navigating grids in new york city through grounded dialogue. 2018.

[57] T. Degris, M. White, and R. S. Sutton. Off-policy actor-critic. *arXiv preprint arXiv :1205.4839*, 2012.

[58] P. Del Moral. *Feynman-Kac Formulae : Genealogical and Interacting Particle Systems With Applications*. Springer, 05 2004. ISBN 0387202684. doi : 10.1007/978-1-4684-9393-1.

[59] P. Del Moral, A. Doucet, and S. S. Singh. A backward particle interpretation of feynman-kac formulae. *ESAIM : Mathematical Modelling and Numerical Analysis*, 44(5) :947–975, 2010.

[60] P. Del Moral, A. Doucet, and S. S. Singh. Uniform stability of a particle approximation of the optimal filter derivative. *SIAM Journal on Control and Optimization*, 53(3) :1278–1304, 2015.

[61] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *jrssb*, 39 :1–38, 1977.

[62] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society : Series B*, 39(1) :1–38, 1977.

[63] R. Deng, Y. Cao, B. Chang, L. Sigal, G. Mori, and M. A. Brubaker. Variational hyper rnn for sequence modeling. *arXiv preprint arXiv :2002.10501*, 2020.

[64] Y. Deng, Y. Kim, J. Chiu, D. Guo, and A. M. Rush. Latent alignment and variational attention. *arXiv preprint arXiv :1807.03756*, 2018.

[65] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT : Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi : 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[66] L. Dinh, J. Sohl-Dickstein, and S. Bengio. Density estimation using real nvp. *arXiv preprint arXiv :1605.08803*, 2016.

[67] R. Douc, A. Garivier, E. Moulines, J. Olsson, et al. Sequential monte carlo smoothing for general state space hidden markov models. *The Annals of Applied Probability*, 21(6) :2109–2145, 2011.

[68] R. Douc, E. Moulines, and D. Stoffer. *Nonlinear time series : theory, methods and applications with R examples*. CRC Press, 2014.

[69] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *Statistics and computing*, 10(3) :197–208, 2000.

[70] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte-Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10 :197–208, 2000.

[71] C. Dubarry and S. Le Corff. Non-asymptotic deviation inequalities for smoothed additive functionals in nonlinear state-space models. *Bernoulli*, 19(5B) :2222–2249, 2013.

[72] C. Dubarry and S. Le Corff. Non-asymptotic deviation inequalities for smoothed additive functionals in nonlinear state-space models. *Bernoulli*, 19(5B) :2222–2249, 2013.

[73] G. Dulac-Arnold, R. Evans, H. van Hasselt, P. Sunehag, T. Lillicrap, J. Hunt, T. Mann, T. Weber, T. Degris, and B. Coppin. Deep reinforcement learning in large discrete action spaces. *arXiv preprint arXiv :1512.07679*, 2015.

[74] J. L. Elman. Finding structure in time. *Cognitive science*, 14(2) :179–211, 1990.

[75] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning, et al. Impala : Scalable distributed deep-rl with importance weighted actor-learner architectures. In *Proc. of International Conference on Machine Learning (ICML)*, 2018.

[76] O. Fabius and J. R. van Amersfoort. Variational recurrent auto-encoders. In *Workshop of the International Conference on Learning Representations (ICLR)*, 2015.

[77] Z. Fan, Z. Wei, S. Wang, Y. Liu, and X.-J. Huang. A reinforcement learning framework for natural question generation using bi-discriminators. In *Proc. of the International Conference on Computational Linguistics (ICCL)*, 2018.

[78] H. I. Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P.-A. Muller. Deep learning for time series classification : a review. *Data Mining and Knowledge Discovery*, 33(4) :917–963, 2019.

[79] P. Fearnhead, O. Papaspiliopoulos, and G. O. Roberts. Particle filters for partially observed diffusions. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 70(4) :755–777, 2008.

[80] P. Fearnhead, O. Papaspiliopoulos, G. Roberts, and A. Stuart. Random weight particle filtering of continuous time stochastic processes. *Journal of the Royal Statistical Society : Series B (Statistical Methodology)*, 72(4) : 497–512, 2010.

[81] P. Fearnhead, D. Wyncoll, and J. Tawn. A sequential smoothing algorithm with linear computational cost. *Biometrika*, 97(2) :447–464, 2010.

[82] P. Fearnhead, D. Wyncoll, and J. Tawn. A sequential smoothing algorithm with linear computational cost. *Biometrika*, 97(2) :447–464, 2010.

[83] P. Fearnhead, K. Latuszynski, G. O. Roberts, and G. Sermaidis. Continuous-time importance sampling : Monte carlo methods which avoid time-discretisation error. *arXiv preprint arXiv :1712.06201*, 2017.

[84] W. Fedus, I. J. Goodfellow, and A. M. Dai. Maskgan : Better text generation via filling in the _____. *CoRR*, abs/1801.07736, 2018. URL http://arxiv.org/abs/1801.07736.

[85] D. A. Ferrucci. Introduction to “this is watson”. *IBM Journal of Research and Development*, 56(3.4) :1–1, 2012.

[86] J. R. Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[87] A. Y. Foong, D. R. Burt, Y. Li, and R. E. Turner. On the expressiveness of approximate inference in bayesian neural networks. *arXiv preprint arXiv :1909.00719*, 2019.

[88] M. Fortunato, C. Blundell, and O. Vinyals. Bayesian recurrent neural networks. *arXiv preprint arXiv :1704.02798*, 2017.

[89] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy networks for exploration. In *Proceedings of the International Conference on Representation Learning (ICLR)*, 2018.

[90] M. Fraccaro, S. K. Sønderby, U. Paquet, and O. Winther. Sequential neural models with stochastic layers. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2016.

[91] M. Freitag and Y. Al-Onaizan. Beam search strategies for neural machine translation. *Proceedings of the First Workshop on Neural Machine Translation*, 2017. doi : 10.18653/v1/w17-3207. URL http://dx.doi.org/10.18653/v1/W17-3207.

[92] S. Fujimoto, H. Hoof, and D. Meger. Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR, 2018.

[93] Y. Gal. Uncertainty in deep learning. *University of Cambridge*, 1(3), 2016.

[94] Y. Gal and Z. Ghahramani. Dropout as a bayesian approximation : Representing model uncertainty in deep learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[95] S. Garg, S. Prabhu, H. Misra, and G. Srinivasaraghavan. Unsupervised contextual paraphrase generation using lexical control and reinforcement learning. *arXiv preprint arXiv :2103.12777*, 2021.

[96] É. Gassiat, A. Cleynen, and S. Robin. Inference in finite state space non parametric hidden markov models and applications. *Statistics and Computing*, 26(1-2) :61–71, 2016.

[97] A. Gatt and E. Krahmer. Survey of the state of the art in natural language generation : Core tasks, applications and evaluation. *Journal of Artificial Intelligence Research*, 61 :65–170, 2018.

[98] S. Gehman, S. Gururangan, M. Sap, Y. Choi, and N. A. Smith. RealToxicityPrompts : Evaluating neural toxic degeneration in language models. In *Findings of the Association for Computational Linguistics : EMNLP 2020*, pages 3356–3369, Online, Nov. 2020. Association for Computational Linguistics. doi : 10.18653/v1/2020.findings-emnlp.301. URL https://aclanthology.org/2020.findings-emnlp.301.

[99] M. Geist and O. Pietquin. Kalman temporal differences. *Journal of artificial intelligence research*, 39 :483–532, 2010.

[100] M. Gerber and N. Chopin. Convergence of sequential quasi-Monte Carlo smoothing algorithms. *Bernoulli*, 23 (4B) :2951–2987, 2017.

[101] M. Ghazvininejad, X. Shi, J. Priyadarshi, and K. Knight. Hafez : an interactive poetry generation system. In *Proceedings of ACL 2017, System Demonstrations*, pages 43–48, Vancouver, Canada, July 2017. Association for Computational Linguistics. URL https://aclanthology.org/P17-4008.

[102] P. Gloaguen, M.-P. Etienne, and S. Le Corff. Online sequential monte carlo smoother for partially observed diffusion processes. *EURASIP Journal on Advances in Signal Processing*, 2018(1) :9, 2018.

[103] P. Gloaguen, S. Le Corff, and J. Olsson. A pseudo-marginal sequential Monte Carlo online smoothing algorithm. Apr. 2021. URL https://hal.archives-ouvertes.fr/hal-02194237. Preprint available online.

[104] S. J. Godsill, A. Doucet, and M. West. Monte carlo smoothing for nonlinear time series. *Journal of the american statistical association*, 99(465) :156–168, 2004.

[105] S. J. Godsill, A. Doucet, and M. West. Monte Carlo smoothing for non-linear time series. *Journal of the American Statistical Association*, 50 :438–449, 2004.

[106] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, and D. Warde-Farley. Sherjil ozair aaron c. courville and yoshua bengio. generative adversarial networks. *CoRR*, 2014.

[107] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, Cambridge, MA, USA, 2016. http://www.deeplearningbook.org.

[108] A. Gopnik and A. Meltzoff. The development of categorization in the second year and its relation to other cognitive and linguistic developments. *Child development*, pages 1523–1531, 1987.

[109] N. Gordon, D. Salmond, and A. Smith. Novel approach to nonlinear/non-Gaussian bayesian state estimation. *Proceedings of the IEEE Conference of Radar Signal Process*, 1993.

[110] N. J. Gordon, D. J. Salmond, and A. F. Smith. Novel approach to nonlinear/non-gaussian bayesian state estimation. In *IEE Proceedings F-radar and signal processing*, volume 140, pages 107–113. IET, 1993.

[111] Y. Goyal, T. Khot, D. Summers-Stay, D. Batra, and D. Parikh. Making the v in vqa matter : Elevating the role of image understanding in visual question answering. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[112] W. Grathwohl, K.-C. Wang, J.-H. Jacobsen, D. Duvenaud, and R. Zemel. Learning the stein discrepancy for training and evaluating energy-based models without sampling. In *International Conference on Machine Learning*, pages 3732–3747. PMLR, 2020.

[113] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 6645–6649. Ieee, 2013.

[114] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic : Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In J. G. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 1856–1865. PMLR, 2018. URL http://proceedings.mlr.press/v80/haarnoja18b.html.

[115] K. Hambardzumyan, H. Khachatrian, and J. May. WARP : Word-level Adversarial ReProgramming. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1 : Long Papers)*, pages 4921–4933, Online, Aug. 2021. Association for Computational Linguistics. doi : 10.18653/v1/2021.acl-long.381. URL https://aclanthology.org/2021.acl-long.381.

[116] N. Hansen. The cma evolution strategy : a comparing review. In *Towards a new evolutionary computation*, pages 75–102. Springer, 2006.

[117] Z. S. Harris. Methods in structural linguistics. 1951.

[118] S. C. Hayes, J. T. Blackledge, and D. Barnes-Holmes. Language and cognition : Constructing an alternative approach within the behavioral tradition. In *Relational Frame Theory*, pages 3–20. Springer, 2002.

[119] J. He, J. Chen, X. He, J. Gao, L. Li, L. Deng, and M. Ostendorf. Deep reinforcement learning with a natural language action space. In *Proc. of Association for Computational Linguistics (ACL)*, 2016.

[120] J. He, D. Spokoyny, G. Neubig, and T. Berg-Kirkpatrick. Lagging inference networks and posterior collapse in variational autoencoders. In *International Conference on Learning Representations (ICLR)*, New Orleans, LA, USA, May 2019. URL https://openreview.net/pdf?id=rylDfnCqF7.

[121] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016.

[122] A. Hening and D. H. Nguyen. Persistence in stochastic lotka–volterra food chains with intraspecific competition. *Bulletin of mathematical biology*, 80(10) :2527–2560, 2018.

[123] J. M. Hernández-Lobato and R. Adams. Probabilistic backpropagation for scalable learning of bayesian neural networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2015.

[124] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9 :1735–1780, 1997.

[125] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8) :1735–1780, 1997.

[126] A. Holtzman, J. Buys, M. Forbes, A. Bosselut, D. Golub, and Y. Choi. Learning to write with cooperative discriminators. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1 : Long Papers)*, pages 1638–1649, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi : 10.18653/v1/P18-1152. URL https://aclanthology.org/P18-1152.

[127] A. Holtzman, J. Buys, L. Du, M. Forbes, and Y. Choi. The curious case of neural text degeneration. *arXiv preprint arXiv :1904.09751*, 2019.

[128] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger. Snapshot ensembles : Train 1, get M for free. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[129] A. Hyvärinen and P. Dayan. Estimation of non-normalized statistical models by score matching. *Journal of Machine Learning Research*, 6(4), 2005.

[130] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for pomdps. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[131] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2016.

[132] N. Jaques, S. Gu, D. Bahdanau, J. M. Hernández-Lobato, R. E. Turner, and D. Eck. Sequence tutor : Conservative fine-tuning of sequence generation models with kl-control. In *Proc. of International Conference on Machine Learning (ICML)*, 2017.

[133] N. Jaques, A. Ghandeharioun, J. H. Shen, C. Ferguson, A. Lapedriza, N. Jones, S. Gu, and R. Picard. Way off-policy batch deep reinforcement learning of implicit human preferences in dialog. *arXiv preprint arXiv :1907.00456*, 2019.

[134] N. Jaques, J. H. Shen, A. Ghandeharioun, C. Ferguson, A. Lapedriza, N. Jones, S. S. Gu, and R. Picard. Human-centric dialog training via offline reinforcement learning. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[135] D. Jin, Z. Jin, J. T. Zhou, and P. Szolovits. Is bert really robust ? natural language attack on text classification and entailment. *arXiv preprint arXiv :1907.11932*, 2, 2019.

[136] J. Johnson, B. Hariharan, L. Van Der Maaten, L. Fei-Fei, C. Lawrence Zitnick, and R. Girshick. Clevr : A diagnostic dataset for compositional language and elementary visual reasoning. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[137] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3128–3137, 2015.

[138] T. Karras, S. Laine, and T. Aila. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4401–4410, 2019.

[139] M. E. Khan, D. Nielsen, V. Tangkaratt, W. Lin, Y. Gal, and A. Srivastava. Fast and scalable bayesian deep learning by weight-perturbation in adam. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[140] Y. Kim, S. Wiseman, and A. M. Rush. A tutorial on deep latent variable models of natural language. *arXiv preprint arXiv :1812.06834*, 2018.

[141] D. P. Kingma and J. Ba. Adam : A method for stochastic optimization. 2014. URL http://arxiv.org/abs/1412.6980. cite arxiv :1412.6980Comment : Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[142] D. P. Kingma and P. Dhariwal. Glow : Generative flow with invertible 1x1 convolutions. *arXiv preprint arXiv :1807.03039*, 2018.

[143] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv :1312.6114*, 2013.

[144] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[145] G. Kitagawa. Monte-Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 1 :1–25, 1996.

[146] G. Kitagawa and S. Sato. Monte carlo smoothing and self-organizing state-space model. In A. Doucet, N. De Freitas, and N. Gordon, editors, *Sequential Monte Carlo methods in Practice*. Springer, 2001.

[147] P. Koehn, F. J. Och, and D. Marcu. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, pages 127–133, 2003. URL https://aclanthology.org/N03-1017.

[148] A. Koenecke, A. Nam, E. Lake, J. Nudell, M. Quartey, Z. Mengesha, C. Toups, J. R. Rickford, D. Jurafsky, and S. Goel. Racial disparities in automated speech recognition. *Proceedings of the National Academy of Sciences*, 117(14) :7684–7689, 2020.

[149] S. Kottur, J. M. F. Moura, D. Parikh, D. Batra, and M. Rohrbach. CLEVR-dialog : A diagnostic dataset for multi-round reasoning in visual dialog. In *Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2019.

[150] B. Krause, A. D. Gotmare, B. McCann, N. S. Keskar, S. Joty, R. Socher, and N. F. Rajani. GeDi : Generative discriminator guided sequence generation. In *Findings of the Association for Computational Linguistics : EMNLP 2021*, pages 4929–4952, Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi : 10.18653/v1/2021.findings-emnlp.424. URL https://aclanthology.org/2021.findings-emnlp.424.

[151] H. J. Kushner and G. G. Yin. *Stochastic Approximation Algorithms and Applications*. Springer, 1997.

[152] B. Lakshminarayanan, A. Pritzel, and C. Blundell. Simple and scalable predictive uncertainty estimation using deep ensembles. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[153] D. Lawson, G. Tucker, C. A. Naesseth, C. J. Maddison, R. P. Adams, and Y. W. Teh. Twisted variational sequential monte carlo. In *Third workshop on Bayesian Deep Learning (NeurIPS)*, 2018.

[154] A. Lazaridou, A. Peysakhovich, and M. Baroni. Multi-agent cooperation and the emergence of (natural) language. *CoRR*, abs/1612.07182, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1612.html#LazaridouPB16b.

[155] A. Lazaridou, K. M. Hermann, K. Tuyls, and S. Clark. Emergence of linguistic communication from referential games with symbolic and pixel input. *arXiv preprint arXiv :1804.03984*, 2018.

[156] A. Lazaridou, A. Potapenko, and O. Tieleman. Multi-agent communication meets natural language : Synergies between functional and structural language learning. In *Proc. of Association for Computational Linguistics (ACL)*, 2020.

[157] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood. Auto-encoding sequential monte carlo. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2018.

[158] F. Le Gland and L. Mevel. Recursive estimation in HMMs. In *Proc. IEEE Conf. Decis. Control*, pages 3468–3473, 1997.

[159] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1 :541–551, 1989.

[160] C. Lee, K. Cho, and W. Kang. Mixout : Effective regularization to finetune large-scale pretrained language models. *arXiv preprint arXiv :1909.11299*, 2019.

[161] G. H. Lee and K. J. Lee. Automatic text summarization using reinforcement learning with embedding features. In *Proc. of the International Joint Conference on Natural Language Processing (IJCNLP)*, 2017.

[162] O. Lemon and O. Pietquin. Machine learning for spoken dialogue systems. In *Proc. of Conference of the International Speech Communication Association (ISCA)*, 2007.

[163] B. Lester, R. Al-Rfou, and N. Constant. The power of scale for parameter-efficient prompt tuning. *CoRR*, abs/2104.08691, 2021. URL https://arxiv.org/abs/2104.08691.

[164] M. Lewis, D. Yarats, Y. N. Dauphin, D. Parikh, and D. Batra. Deal or no deal ? end-to-end learning for negotiation dialogues. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2017.

[165] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART : Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics. doi : 10.18653/v1/2020.acl-main.703. URL https://aclanthology.org/2020.acl-main.703.

[166] B. Li, J. He, G. Neubig, T. Berg-Kirkpatrick, and Y. Yang. A surprisingly effective fix for deep latent variable modeling of text. *arXiv preprint arXiv :1909.00868*, 2019.

[167] C. Li, X. Gao, Y. Li, B. Peng, X. Li, Y. Zhang, and J. Gao. Optimus : Organizing sentences via pre-trained modeling of a latent space. *arXiv preprint arXiv :2004.04092*, 2020.

[168] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan. A diversity-promoting objective function for neural conversation models. In *Proc. of the North American Chapter of the Association for Computational Linguistics (NAACL)*, 2015.

[169] J. Li, W. Monroe, A. Ritter, M. Galley, J. Gao, and D. Jurafsky. Deep reinforcement learning for dialogue generation. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2016.

[170] J. Li, T. Tang, W. X. Zhao, and J.-R. Wen. Pretrained language model for text generation : A survey. In Z.-H. Zhou, editor, *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pages 4492–4499. International Joint Conferences on Artificial Intelligence Organization, 8 2021. doi : 10.24963/ijcai.2021/612. URL https://doi.org/10.24963/ijcai.2021/612. Survey Track.

[171] S. Li, X. Jin, Y. Xuan, X. Zhou, W. Chen, Y.-X. Wang, and X. Yan. Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting. In *Proceedings of the Advances in Neural Information Processing Systems*, 2019.

[172] X. L. Li and P. Liang. Prefix-tuning : Optimizing continuous prompts for generation. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1 : Long Papers)*, pages 4582–4597, Online, Aug. 2021. Association for Computational Linguistics. doi : 10.18653/v1/2021.acl-long.353. URL https://aclanthology.org/2021.acl-long.353.

[173] Z. Li, X. Jiang, L. Shang, and H. Li. Paraphrase generation with deep reinforcement learning. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2018.

[174] B. Lim, S. O. Arik, N. Loeff, and T. Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *arXiv preprint arXiv :1912.09363*, 2019.

[175] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco : Common objects in context. In *Proc. of European Conference on Computer Vision (ECCV)*, 2014.

[176] Z. Lin, M. Feng, C. N. dos Santos, M. Yu, B. Xiang, B. Zhou, and Y. Bengio. A structured self-attentive sentence embedding. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2014.

[177] Z. Lin, G. I. Winata, P. Xu, Z. Liu, and P. Fung. Variational transformers for diverse response generation. *arXiv preprint arXiv :2003.12738*, 2020.

[178] J. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93 :1032–1044, 1998.

[179] J. S. Liu and J. S. Liu. *Monte Carlo strategies in scientific computing*, volume 10. Springer, 2001.

[180] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv :1801.10198*, 2018.

[181] S. Liu, N. Yang, M. Li, and M. Zhou. A recursive recurrent neural network for statistical machine translation. 2014.

[182] X. Liu, Y. Zheng, Z. Du, M. Ding, Y. Qian, Z. Yang, and J. Tang. GPT understands, too. *CoRR*, abs/2103.10385, 2021. URL https://arxiv.org/abs/2103.10385.

[183] J. Lu, D. Batra, D. Parikh, and S. Lee. Vilbert : Pretraining task-agnostic visiolinguistic representations for vision-and-language tasks. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[184] J. Lu, V. Goswami, M. Rohrbach, D. Parikh, and S. Lee. 12-in-1 : Multi-task vision and language representation learning. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[185] Y. Lu, S. Singhal, F. Strub, O. Pietquin, and A. Courville. Countering language drift with seeded iterated learning. In *Proc. of International Conference on Machine Learning (ICML)*, 2020.

[186] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of research and development*, 1(4) :309–317, 1957.

[187] J. Luketina, N. Nardelli, G. Farquhar, J. Foerster, J. Andreas, E. Grefenstette, S. Whiteson, and T. Rocktäschel. A survey of reinforcement learning informed by natural language. In *Proc. of International Joint Conferences on Artificial Intelligence (IJCAI)*, 2019.

[188] B. Lütjens, M. Everett, and J. P. How. Safe reinforcement learning with model uncertainty estimates. *2019 International Conference on Robotics and Automation (ICRA)*, pages 8662–8668, 2019.

[189] X. Ma, P. Karkus, D. Hsu, and W. S. Lee. Particle filter recurrent neural networks. pages 5101–5108, 2020. URL https://aaai.org/ojs/index.php/AAAI/article/view/5952.

[190] D. J. MacKay. A practical bayesian framework for backpropagation networks. *Neural computation*, 4(3) : 448–472, 1992.

[191] C. J. Maddison, J. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. Teh. Filtering variational objectives. In *Proceedings of Advances in Neural Information Processing Systems (NIPS)*, 2017.

[192] J. S. Martin, A. Jasra, S. S. Singh, N. Whiteley, P. Del Moral, and E. McCoy. Approximate Bayesian Computation for smoothing. *Stochastic Analysis and Applications*, 32 :397–420, 2014.

[193] A. Martins and R. Astudillo. From softmax to sparsemax : A sparse model of attention and multi-label classification. In M. F. Balcan and K. Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1614–1623, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/martins16.html.

[194] T. Michelot, R. Langrock, and T. A. Patterson. movehmm : an r package for the statistical modelling of animal movement data using hidden markov models. *Methods in Ecology and Evolution*, 7(11) :1308–1315, 2016.

[195] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*, 2010.

[196] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540) : 529–533, Feb. 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

[197] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Thirty-second AAAI conference on artificial intelligence*, 2018.

[198] A. K. Moretti, Z. Wang, L. Wu, I. Drori, and I. Pe'er. Particle smoothing variational objectives. *arXiv preprint arXiv :1909.09734*, 2019.

[199] M. Mosbach, M. Andriushchenko, and D. Klakow. On the stability of fine-tuning bert : Misconceptions, explanations, and strong baselines. *arXiv preprint arXiv :2006.04884*, 2020.

[200] N. Mostafazadeh, I. Misra, J. Devlin, M. Mitchell, X. He, and L. Vanderwende. Generating natural questions about an image. In *Proc. of Association for Computational Linguistics (ACL)*, 2016.

[201] M. C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Systems*, 3, 1989.

[202] R. Munos, T. Stepleton, A. Harutyunyan, and M. G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Proc. of Neural Information Processing Systems (NIPS)*, 2016.

[203] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei. Variational sequential monte carlo. In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2018.

[204] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Icml*, 2010.

[205] K. Narasimhan, T. Kulkarni, and R. Barzilay. Language understanding for text-based games using deep reinforcement learning. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2015.

[206] R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012.

[207] T. Nguyen, S. Le Corff, and É. Moulines. On the two-filter approximations of marginal smoothing distributions in general state-space models. *Advances in Applied Probability*, 50(1) :154–177, 2017.

[208] E. P. Odum and G. W. Barrett. *Fundamentals of ecology*, volume 3. Saunders Philadelphia, 1971.

[209] J. Olsson and J. W. Alenlöv. Particle-based online estimation of tangent filters with application to parameter estimation in nonlinear state-space models. *Annals of the Institute of Statistical Mathematics*, 72(2) :545–576, 2020.

[210] J. Olsson, O. Cappe, R. Douc, and E. Moulines. Sequential monte carlo smoothing with application to parameter estimation in nonlinear state space models. *Bernoulli*, 14(1) :155–179, 2008.

[211] J. Olsson, J. Ströjby, et al. Particle-based likelihood inference in partially observed diffusion processes using generalised poisson estimators. *Electronic Journal of Statistics*, 5 :1090–1122, 2011.

[212] J. Olsson, J. Westerborn, et al. Efficient particle-based online smoothing in general hidden Markov models : the PaRIS algorithm. *Bernoulli*, 23(3) :1951–1996, 2017.

[213] I. Osband, C. Blundell, A. Pritzel, and B. Van Roy. Deep exploration via bootstrapped dqn. *Advances in neural information processing systems*, 29 :4026–4034, 2016.

[214] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. Bleu : a method for automatic evaluation of machine translation. In *Proc. of the Association for Computational Linguistics (ACL)*, 2002.

[215] E. Parisotto, F. Song, J. Rae, R. Pascanu, C. Gulcehre, S. Jayakumar, M. Jaderberg, R. L. Kaufman, A. Clark, S. Noury, et al. Stabilizing transformers for reinforcement learning. In *International Conference on Machine Learning*, pages 7487–7498. PMLR, 2020.

[216] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012. URL http://arxiv.org/abs/1211.5063.

[217] R. Pasunuru and M. Bansal. Reinforced video captioning with entailment rewards. *arXiv preprint arXiv :1708.02300*, 2017.

[218] C. Patil and M. Patwardhan. Visual question generation : The state of the art. *ACM Computing Surveys (CSUR)*, 53(3) :1–22, 2020.

[219] R. Paulus, C. Xiong, and R. Socher. A deep reinforced model for abstractive summarization. In *Proc. of the International Conference on Learning Representations (ICLR)*, 2017.

[220] T. Pearce, A. Brintrup, M. Zaki, and A. Neely. High-quality prediction intervals for deep learning : A distribution-free, ensembled approach. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[221] J. Pennington, R. Socher, and C. D. Manning. Glove : Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[222] M. E. Peters, S. Ruder, and N. A. Smith. To tune or not to tune ? adapting pretrained representations to diverse tasks. *arXiv preprint arXiv :1903.05987*, 2019.

[223] O. Pietquin and T. Dutoit. A probabilistic framework for dialog simulation and optimal strategy learning. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(2) :589–599, 2006.

[224] M. K. Pitt and N. Shephard. Filtering via simulation : Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446) :590–599, 1999.

[225] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4) :838–855, 1992.

[226] G. Poyiadjis, A. Doucet, and S. Singh. Particle approximations of the score and observed information matrix in state space models with application to parameter estimation. *Biometrika*, 98 :65–80, 2011.

[227] G. Qin and J. Eisner. Learning how to ask : Querying LMs with mixtures of soft prompts. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 5203–5212, Online, June 2021. Association for Computational Linguistics. doi : 10.18653/v1/2021.naacl-main.410. URL https://aclanthology.org/2021.naacl-main.410.

[228] L. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[229] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *PROCEEDINGS OF THE IEEE*, pages 257–286, 1989.

[230] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. 2018.

[231] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre-training. urlhttps ://openai.com/blog/language-unsupervised/, 2018.

[232] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8) :9, 2019.

[233] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv :1910.10683*, 2019.

[234] M. Ranzato, S. Chopra, M. Auli, and W. Zaremba. Sequence level training with recurrent neural networks. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[235] A. Razavi, A. van den Oord, and O. Vinyals. Generating diverse high-fidelity images with vq-vae-2. In *Advances in neural information processing systems*, pages 14866–14876, 2019.

[236] D. R. Reddy et al. Speech understanding systems : A summary of results of the five-year research effort. department of computer science, 1977.

[237] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn : Towards real-time object detection with region proposal networks. In *Proc. of Neural Information Processing Systems (NIPS)*, 2015.

[238] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel. Self-critical sequence training for image captioning. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[239] T. Robinson, M. Hochberg, and S. Renals. The use of recurrent neural networks in continuous speech recognition. In *Automatic speech and speaker recognition*, pages 233–258. Springer, 1996.

[240] D. B. Rubin. Comment : A noniterative sampling/importance resampling alternative to the data augmentation algorithm for creating a few imputations when fractions of missing information are modest : The sir algorithm, June 1987. URL http://wrap.warwick.ac.uk/24938/.

[241] D. B. Rubin. Using the sir algorithm to simulate posterior distributions. *Bayesian statistics*, 3 :395–402, 1988.

[242] S. Ruder. *Neural transfer learning for natural language processing*. PhD thesis, NUI Galway, 2019.

[243] A. Saleh, N. Jaques, A. Ghandeharioun, J. Shen, and R. Picard. Hierarchical reinforcement learning for open-domain dialog. In *Proc. of Conference on Artificial Intelligence (AAAI)*, 2020.

[244] D. Salinas, V. Flunkert, J. Gasthaus, and T. Januschowski. Deepar : Probabilistic forecasting with autoregressive recurrent networks. *International Journal of Forecasting*, 36(3) :1181–1191, 2020.

[245] S. Särkkä. *Bayesian Filtering and Smoothing*. Cambridge University Press, New York, NY, USA, 2013.

[246] S. Särkkä, A. Vehtari, and J. Lampinen. Rao-Blackwellized particle filter for multiple target tracking. *Inofrmation Fusion*, 8(1) :2–15, 2007.

[247] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[248] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[249] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

[250] P. Schwaller, T. Laino, T. Gaudin, P. Bolgar, C. A. Hunter, C. Bekas, and A. A. Lee. Molecular transformer : A model for uncertainty-calibrated chemical reaction prediction. *ACS central science*, 5(9) :1572–1583, 2019.

[251] S. Semeniuta, A. Severyn, and S. Gelly. On accurate evaluation of gans for language generation. *arXiv preprint arXiv :1806.04936*, 2018.

[252] R. Sennrich, B. Haddow, and A. Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv :1508.07909*, 2015.

[253] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012.

[254] M. Seurin, P. Preux, and O. Pietquin. " i'm sorry dave, i'm afraid i can't do that" deep q-learning from forbidden actions. In *Proc. of the International Joint Conference on Neural Networks (IJCNN)*, 2020.

[255] S. Shankar and S. Sarawagi. Posterior attention models for sequence to sequence learning. In *International Conference on Learning Representations*, 2018.

[256] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. AutoPrompt : Eliciting Knowledge from Language Models with Automatically Generated Prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 4222–4235, Online, Nov. 2020. Association for

Computational Linguistics. doi : 10.18653/v1/2020.emnlp-main.346. URL https://aclanthology.org/2020.emnlp-main.346.

[257] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, page 103535, 2021.

[258] S. Singh, D. Litman, M. Kearns, and M. Walker. Optimizing dialogue management with reinforcement learning : Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16 :105–133, 2002.

[259] H. Song, D. Rajan, J. J. Thiagarajan, and A. Spanias. Attend and diagnose : Clinical time series analysis using attention models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2018.

[260] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout : A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56) :1929–1958, 2014. URL http://jmlr.org/papers/v15/srivastava14a.html.

[261] N. Stiennon, L. Ouyang, J. Wu, D. M. Ziegler, R. Lowe, C. Voss, A. Radford, D. Amodei, and P. Christiano. Learning to summarize from human feedback. *arXiv preprint arXiv :2009.01325*, 2020.

[262] F. Strub, H. De Vries, J. Mary, B. Piot, A. Courville, and O. Pietquin. End-to-end optimization of goal-driven and visually grounded dialogue systems. In *Proc. of International Joint Conferences on Artificial Intelligence (IJCAI)*, 2017.

[263] M. Stubbs. *Text and corpus analysis : Computer-assisted studies of language and culture*. Blackwell Oxford, 1996.

[264] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *ICML*, 2011.

[265] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[266] R. S. Sutton and A. G. Barto. *Reinforcement Learning : An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[267] R. S. Sutton, A. G. Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[268] R. S. Sutton, D. A. McAllester, S. P. Singh, Y. Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *Proc. of Neural Information Processing Systems (NIPS)*, 1999.

[269] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[270] V. Tadić. Analyticity, convergence, and convergence rate of recursive maximum-likelihood estimation in hidden Markov models. *IEEE Transactions on Information Theory*, 56 :6406–6432, 2010.

[271] N. Tagasovska and D. Lopez-Paz. Single-model uncertainties for deep learning. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[272] B. Tan, Z. Yang, M. Al-Shedivat, E. Xing, and Z. Hu. Progressive generation of long text with pretrained language models. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics : Human Language Technologies*, pages 4313–4324, Online, June 2021. Association for Computational Linguistics. doi : 10.18653/v1/2021.naacl-main.341. URL https://aclanthology.org/2021.naacl-main.341.

[273] G. Tennenholtz and S. Mannor. The natural language of actions. In *Proc. of International Conference on Machine Learning (ICML)*, 2019.

[274] M. Teye, H. Azizpour, and K. Smith. Bayesian uncertainty estimation for batch normalized deep networks. In *Proceedings of the International Conference on Machine Learning (ICML)*, 2018.

[275] T. Tran, T. Do, I. D. Reid, and G. Carneiro. Bayesian generative active deep learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6295–6304. PMLR, 2019. URL http://proceedings.mlr.press/v97/tran19a.html.

[276] J. N. Tsitsiklis and B. Van Roy. Analysis of temporal-diffference learning with function approximation. In *Advances in neural information processing systems*, pages 1075–1081, 1997.

[277] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[278] R. Vedantam, C. Lawrence Zitnick, and D. Parikh. Cider : Consensus-based image description evaluation. In *Proc. of Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[279] E. Wallace, S. Feng, N. Kandpal, M. Gardner, and S. Singh. Universal adversarial triggers for attacking and analyzing nlp, 2021.

[280] B. Wang, T. Li, Z. Yan, G. Zhang, and J. Lu. Deeppipe : A distribution-free uncertainty quantification approach for time series forecasting. *Neurocomputing*, 2020.

[281] P.-H. Wang, S.-I. Hsieh, S.-C. Chang, Y.-T. Chen, J.-Y. Pan, W. Wei, and D.-C. Juan. Contextual temperature for language modeling. *arXiv preprint arXiv :2012.13575*, 2020.

[282] X. Wang, E. Lebarbier, J. Aubert, and S. Robin. Variational inference for coupled hidden markov models applied to the joint detection of copy number variations. *The International Journal of Biostatistics*, 15, 06 2017. doi : 10.1515/ijb-2018-0023.

[283] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://dblp.uni-trier.de/db/journals/corr/corr1611.html#WangBHMMKF16.

[284] Z. Wang, V. Bapst, N. Heess, V. Mnih, R. Munos, K. Kavukcuoglu, and N. de Freitas. Sample efficient actor-critic with experience replay. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[285] R. Wen, K. Torkkola, B. Narayanaswamy, and D. Madeka. A multi-horizon quantile recurrent forecaster. In *Proceedings of the Advances in Neural Information Processing Systems (NIPS)*, 2017.

[286] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4) :229–256, 1992.

[287] R. A. Wilson and L. Foglia. Embodied cognition. 2011.

[288] C. Wu, P. Z. Wang, and W. Y. Wang. On the encoder-decoder incompatibility in variational text modeling and beyond. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3449–3464, Online, July 2020. Association for Computational Linguistics. doi : 10.18653/v1/2020.acl-main.316. URL https://aclanthology.org/2020.acl-main.316.

[289] N. Wu, B. Green, X. Ben, and S. O'Banion. Deep transformer models for time series forecasting : The influenza prevalence case. *arXiv preprint arXiv :2001.08317*, 2020.

[290] Q. Wu, Y. Zhang, Y. Li, and Z. Yu. Alternating roles dialog model with large-scale pre-trained language models. In *Proc. of European Association for Computational Linguistics (EACL)*, 2019.

[291] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, et al. Google's neural machine translation system : Bridging the gap between human and machine translation. *arXiv preprint arXiv :1609.08144*, 2016.

[292] Y. Xiao and W. Y. Wang. Quantifying uncertainties in natural language processing tasks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[293] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell : Neural image caption generation with visual attention. In *Proc. of International conference on machine learning (ICML)*, 2015.

[294] L. C. Yan, B. Yoshua, and H. Geoffrey. Deep learning. *nature*, 521(7553) :436–444, 2015.

[295] J. Yang, J. Lu, S. Lee, D. Batra, and D. Parikh. Visual curiosity : Learning to ask questions to learn visual recognition. In *Proc. of International Conference on Machine Learning (ICML)*, 2018.

[296] S. Yao, R. Rao, M. Hausknecht, and K. Narasimhan. Keep calm and explore : Language models for action generation in text-based games. In *Proc. of Empirical Methods in Natural Language Processing (EMNLP)*, 2020.

[297] C. Yau, O. Papaspiliopoulos, G. O. Roberts, and C. Holmes. Bayesian non-parametric hidden Markov models with applications in genomics. *jrssb*, 73 :1–21, 2011.

[298] S. Yonekura and A. Beskos. Online smoothing for diffusion processes observed with noise. 2020. ArXiv :2003.12247.

[299] S. Young. Using pomdps for dialog management. In *IEEE Spoken Language Technology Workshop*, 2006.

[300] T. Zahavy, M. Haroush, N. Merlis, D. J. Mankowitz, and S. Mannor. Learn what not to learn : Action elimination with deep reinforcement learning. In *Proc. of Neural Information Processing Systems (NeurIPS)*, 2018.

[301] H. Zhang, D. Duckworth, D. Ippolito, and A. Neelakantan. Trading off diversity and quality in natural language generation. In *Proc. of European Association for Computational Linguistics (EACL)*, 2020.

[302] H. Zhang, A. X. Lu, M. Abdalla, M. McDermott, and M. Ghassemi. Hurtful words : quantifying biases in clinical contextual word embeddings. In *proceedings of the ACM Conference on Health, Inference, and Learning*, pages 110–120, 2020.

[303] J. O. Zhang, A. Sax, A. Zamir, L. J. Guibas, and J. Malik. Side-tuning : Network adaptation via additive side networks. 2019.

[304] X. Zhang, F. X. Yu, S. Karaman, W. Zhang, and S.-F. Chang. Heated-up softmax embedding. *arXiv preprint arXiv :1809.04157*, 2018.

[305] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin. Adversarial feature matching for text generation. In *Proc. of International Conference on Machine Learning (ICML)*, 2017.

[306] L. Zhu and N. Laptev. Deep and confident prediction for time series at uber. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW)*, 2017.

[307] Y. Zhu, S. Lu, L. Zheng, J. Guo, W. Zhang, J. Wang, and Y. Yu. Texygen : A benchmarking platform for text generation models. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*, pages 1097–1100, 2018.

[308] D. M. Ziegler, N. Stiennon, J. Wu, T. B. Brown, A. Radford, D. Amodei, P. Christiano, and G. Irving. Fine-tuning language models from human preferences. *arXiv preprint arXiv :1909.08593*, 2019.

[309] W. Zucchini, I. Mac Donald, and R. Langrock. *Hidden Markov models for time series : an introduction using R*. CRC Press, 2017.

ECOLE
DOCTORALE
DE MATHEMATIQUES
HADAMARD

Titre : Modèles et algorithmes d'apprentissage profond pour de la donnée séquentielle - Applications aux modèles de langage et à la quantification d'incertitude

Mots clés : modèles génératifs profonds, méthodes de Monte Carlo Séquentielles, lissage en ligne, modèles de langage, apprentissage par renforcement, quantification d'incertitude

Résumé : Dans ce manuscrit, nous proposons de nouveaux algorithmes et modèles pour résoudre les problèmes d'apprentissage profond sur de la donnée séquentielle. Ces contributions sont motivées en partie par les problématiques posées par l'apprentissage de modèles de langage basés sur des réseaux de neurones. Un premier axe de recherche développe un nouveau modèle génératif profond à données latentes inspiré de l'architecture du Transformer, et entraîné grâce des méthodes de Monte Carlo Séquentielles (SMC). Celui-ci permet de mieux modéliser la diversité du langage, et fournit un cadre afin de quantifier l'incertitude pour des problèmes de régression séquentiels. Un deuxième axe de recherche vise à faciliter l'utilisation de techniques de type SMC dans le cadre de l'apprentissage profond, en développant un nouvel algorithme de lissage à coût computationnel largement réduit. Celui s'applique aussi à un éventail large de modèles à espace d'états, ceux pour lesquels la densité de transition ou la distribution conditionnelle des observations est inconnue, donc en particulier aux modèles génératifs profonds. Finalement, un troisième axe de recherche propose le premier algorithme d'apprentissage par renforcement permettant d'apprendre des modèles de langage conditionnels "ex-nihilo" (i.e sans jeux de données supervisés), basé sur un mécanisme de troncation de l'espace d'actions par un modèle de langage pré-entraîné.

Title : Deep Learning models and algorithms for sequential data problems - Applications to Language Modelling and Uncertainty Quantification

Keywords : Deep Generative Models, Sequential Monte Carlo, online smoothing, Language Models, Reinforcement Learning, uncertainty quantification

Abstract : In this thesis, we develop new models and algorithms to solve deep learning tasks on sequential data problems, with the perspective of tackling the pitfalls of current approaches for learning language models based on neural networks. A first research work develops a new deep generative model for sequential data based on a Transformer architecture and trained using Sequential Monte Carlo Methods. The approach enables to better model diversity in language modelling tasks, and better quantify uncertainty in sequential regression problems. A second research work aims to facilitate the use of SMC techniques within deep learning architectures, by developing a new online smoothing algorithm with reduced computational cost. The algorithm is also applicable on a wider scope of state-space models, the ones for which the transition density or conditional distribution of the observations is intractable, hence is usable on deep generative models. Finally, a third research work proposes the first reinforcement learning that enables to learn conditional language models from scratch (i.e without supervised datasets), based on a truncation mechanism of the natural language action space with a pretrained language model.

Institut Polytechnique de Paris

91120 Palaiseau, France