Deep generative neural networks for novelty generation : a foundational framework, metrics and experiments

Mehdi Cherti

NNT : 2018SACLS029

Thèse de doctorat

Deep generative neural networks for novelty generation: a foundational framework, metrics and experiments

Thèse de doctorat de l'Université Paris-Saclay
préparée à Université Paris-Sud

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 26/01/18, par

M. MEHDI CHERTI

Composition du Jury :

M. Achille Stocchi Directeur, Laboratoire de l'accélérateur Lineaire (LAL)	Président
M. Eswaran Subrahmanian Research professor, Carnegie Mellon University (CMU)	Rapporteur
M. Douglas Eck Research scientist, Google Brain	Rapporteur
M. Ludovic Denoyer Professeur, Université Pierre et Marie Curie, LIP6	Examinateur
Mme Cécile Germain Professeur, Université Paris-Sud	Examinatrice
M. Armand Hatchuel Professeur, Mines Paristech, Centre de gestion (CGS)	Examinateur
M. Balázs Kégl Directeur de recherche, Laboratoire de l'accélérateur Linéaire (LAL)	Directeur de thèse
M. Akın Kazakçı Professeur assistant, Mines Paristech, Centre de gestion(CGS)	Co-Directeur de thèse

LAL 18-003

Remerciements

Merci Balàzs pour toute ton aide, ta patience avec moi et ta confiance en mes aptitudes, je te suis extrêmement reconnaissant. Tu m'as appris énormément de choses durant la thèse, que ce soit l'aspect technique ou l'aspect "meta" sur comment faire de la recherche, je te remercie parce j'ai énormément progressé grâce à toi.

Merci Akın de toujours croire en moi et de toujours me diriger vers des problématiques importantes. Grâce à toi, j'ai appris énormément de choses, merci de m'avoir transmis ton savoir, y compris des choses auxquelles je n'aurai peut être jamais pensé à étudier, merci beaucoup.

Balàzs, Akın, j'apprécie tous les deux votre passion et votre motivation qui me poussent à toujours faire mieux.

Je remercie également tous les membres du jury d'avoir accepté d'évaluer ma thèse. Merci au staff du LAL, je remercie particulièrement Sylvie Prandt et Valentine Marcuola. Je tiens à également remercier Stéphanie Druetta à l'école doctorale.

Merci à tous mes amis de m'avoir accompagné durant cette thèse et m'avoir aidé à sortir un peu de mon cocon. Merci beaucoup Aram pour ta générosité, ta disponibilité et de m'avoir aidé durant des moments critiques, je te suis reconnaissant. Merci à Biro, Boss et Ismail pour votre gentillesse et vos délires. Merci Boss de m'avoir aidé durant mon premier mois en France. Merci Thomas, j'ai apprécié toutes les discussions qu'on a eu et les moments qu'on a passés ensemble, ça m'a fait beaucoup de bien.

Merci également à tous mes collègues ou anciens collègues au LAL et amis qui m'ont accompagné durant cette thèse: Alex, Darko, Diego, Djalel, Ema, François-David, Sourava, Victoria et Yetkin.

Merci Zak et Mélanie pour votre gentillesse et pour m'avoir aidé durant mes premiers mois en France, je vous suis très reconnaissant.

Merci énormément c mes grands parents pour leur bonté et leur bienveillance à mon égard.

Merci Achraf et Adam pour vos encouragements tout au long de ma thèse.

Merci maman et papa pour votre soutien constant, je n'aurai jamais pu faire ce chemin sans vous. Je vous serai toujours reconnaissant. Merci d'avoir toujours été présents et d'avoir toujours pensé à mon bien. Cette thèse vous est dédiée.

Résumé

Des avancées significatives sur les réseaux de neurones profonds ont récemment permis le développement de technologies importantes comme les voitures autonomes et les assistants personnels intelligents basés sur la commande vocale. La plupart des succès en apprentissage profond concernent la prédiction, alors que les percées initiales viennent des modèles génératifs. Actuellement, même s'il existe des outils puissants dans la littérature des modèles génératifs basés sur les réseaux profonds, ces techniques sont essentiellement utilisées pour la prédiction ou pour générer des objets *connus* (i.e., des images de haute qualité qui appartiennent à des classes connues) : un objet généré qui est *à priori* inconnu est considéré comme une erreur (Salimans et al., 2016) ou comme un objet fallacieux (Bengio et al., 2013b). En d'autres termes, quand la prédiction est considérée comme le seul objectif possible, la nouveauté est vue comme une erreur - que les chercheurs ont essayé d'éliminer au maximum. Cette thèse défends le point de vue que, plutôt que d'éliminer ces *nouveautés*, on devrait les étudier et étudier le potentiel génératif des réseaux neuronaux pour créer de la nouveauté utile - particulièrement sachant l'importance économique et sociétale de la création d'objets nouveaux dans les sociétés contemporaines. Cette thèse a pour objectif d'étudier la génération de la nouveauté et sa relation avec les modèles de connaissance produits par les réseaux neurones profonds génératifs. Notre première contribution est la démonstration de l'importance des représentations et leur impact sur le type de nouveautés qui peuvent être générées : une conséquence clé est qu'un agent créatif a besoin de rereprésenter les objets connus et utiliser cette représentation pour générer des objets nouveaux. Ensuite, on démontre que les fonctions objectives traditionnelles utilisées dans la théorie de l'apprentissage statistique, comme le maximum de vraisemblance, ne sont pas nécessairement les plus adaptées pour étudier la génération de nouveauté. On propose plusieurs alternatives à un niveau conceptuel. Un deuxième résultat clé est la confirmation que les modèles actuels - qui utilisent les fonctions objectives traditionnelles - peuvent en effet générer des objets inconnus. Cela montre que même si les fonctions objectives comme le maximum de vraisemblance s'efforcent à éliminer la nouveauté, les implémentations en pratique échouent à le faire. A travers une série d'expérimentations, on étudie le comportement de ces modèles ainsi que les objets qu'ils génèrent. En particulier, on propose une nouvelle tâche et des métriques pour la sélection de bons modèles génératifs pour la génération de la nouveauté. Finalement, la thèse conclue avec une série d'expérimentations qui clarifie les caractéristiques des modèles qui génèrent de la nouveauté. Les expériences montrent que la sparsité, le niveaux du niveau de corruption et la restriction de la

capacité des modèles tuent la nouveauté et que les modèles qui arrivent à reconnaître des objets nouveaux arrivent généralement aussi à générer de la nouveauté.

Abstract

In recent years, significant advances made in deep neural networks enabled the creation of groundbreaking technologies such as self-driving cars and voice-enabled personal assistants. Almost all successes of deep neural networks are about prediction, whereas the initial breakthroughs came from generative models. Today, although we have very powerful deep generative modeling techniques, these techniques are essentially being used for prediction or for generating *known* objects (i.e., good quality images of known classes): any generated object that is *a priori* unknown is considered as a failure mode (Salimans et al., 2016) or as spurious (Bengio et al., 2013b). In other words, when prediction seems to be the only possible objective, novelty is seen as an error that researchers have been trying hard to eliminate. This thesis defends the point of view that, instead of trying to eliminate these *novelties*, we should study them and the generative potential of deep nets to create useful novelty, especially given the economic and societal importance of creating new objects in contemporary societies. The thesis sets out to study novelty generation in relationship with data-driven knowledge models produced by deep generative neural networks. Our first key contribution is the clarification of the importance of representations and their impact on the kind of novelties that can be generated: a key consequence is that a creative agent might need to re-represent known objects to access various kinds of novelty. We then demonstrate that traditional objective functions of statistical learning theory, such as maximum likelihood, are not necessarily the best theoretical framework for studying novelty generation. We propose several other alternatives at the conceptual level. A second key result is the confirmation that current models, with traditional objective functions, can indeed generate unknown objects. This also shows that even though objectives like maximum likelihood are designed to eliminate novelty, practical implementations do generate novelty. Through a series of experiments, we study the behavior of these models and the novelty they generate. In particular, we propose a new task setup and metrics for selecting good generative models. Finally, the thesis concludes with a series of experiments clarifying the characteristics of models that can exhibit novelty. Experiments show that sparsity, noise level, and restricting the capacity of the net eliminates novelty and that models that are better at recognizing novelty are also good at generating novelty.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Artificial intelligence (AI) systems are seen as tools that will profoundly change human society through automation, reduced costs, and the possibility of performing tasks that humans excel at (e.g., pattern recognition, control, abstraction) but also tasks that are beyond human capability (e.g., weather prediction). AI as a field is broad and it includes several subfields, among them we have planning, learning, perception, control (e.g., motion), communication (e.g., through natural language), constraint satisfaction.

Currently, due to its empirical successes, machine learning is considered as the flagship of AI, where almost all successful applications concern prediction. For several well-known practical purposes and within an engineering paradigm, the prediction tasks are extremely valuable, since the predictive paradigm enables access to groundbreaking technologies such as the first generation of industrial autonomous vehicles or voice-enabled personal assistants (e.g., Siri, Amazon Echo). On the research side, it is *safe* to work on prediction methods since it is based on a clearly defined and well-understood framework of statistical learning theory, which allows both an understanding, but also a means for comparing methods and measuring progress.

However, current AI models based on prediction ignore and fail to reproduce several other essential traits of human intelligence, such as emotions, intuitive physics, intuitive psychology, the ability to learn with very few examples (Lake et al., 2017). In failing to consider these essential human cognitive traits, machine learning becomes of little interest for studying experimentally an AI system that can go beyond a simple (however powerful) learning-based paradigm. More importantly, achieving more advanced levels of AI able to reproduce these characteristics seem highly unlikely, simply because the machine learning community overlooks these problems and remains focused on prediction methods.

One such characteristics of human intelligence, at the core of this research, is the ability of humans to create novelty. Humans' ability to create novelty manifests itself in many domains, such as the design of new products (e.g., in the engineering industry), art (e.g., paintings, music), and intellectual works (e.g., scientific theories, novels). While the impact and the relevance of generating novelty has been ascertained and emphasized by and large in many scientific literatures (cognitive psychology, innovation management, design theory), artificial intelligence in general, and machine learning in particular, has remained quite insensitive to the topic.

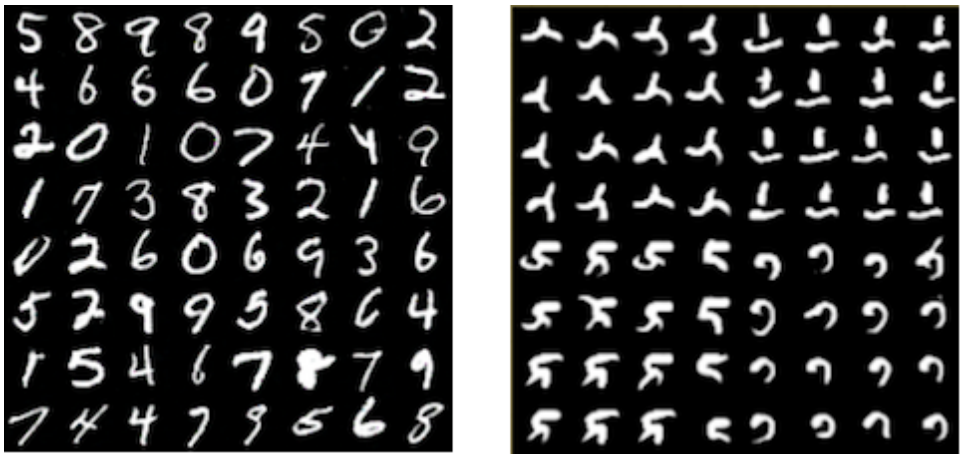
Possibly, the main reason for this lack of interest is the absence of a theoretical framework clearly defining what novelty generation is and what are the tasks where novelty generation with machine learning methods would be useful. Such a framework would ensure subsequent productivity of the research on the topic.

The central goal of this thesis is to contribute to the building of such a framework by laying down the groundwork based on which generation of novelty can be systematically studied within the machine learning field. The starting point and the initial motivation behind this research project were generative models in machine learning. While most of the literature on machine learning is about prediction (and discriminative models), the same literature produced generative models, which are models that can *generate* data. However, currently, most work involving generative models use them to improve prediction models. Moreover, and as we will show throughout the thesis, even when they are used for generation, current generative models are mostly trained to generate *known* objects, making them practically limited for generating *genuine* novelty, They are thus useless for understanding the mechanism by which novelty generation occurs.

What do we know about the "full potential" of generative models? Figure 1.1a and Figure 1.1b show samples generated from current generative models used in the machine learning literature. Which images would be more appropriately qualified as "novel" ? Under the current paradigm, the images at the left side (the digits and the dog) would be considered as valuable since they are reproducing with great details known objects, while the others would be considered as failure modes (Salimans et al., 2016) or as "spurious" (Bengio et al., 2013b). In other terms, when prediction seems to be the only possible objective, novelty is seen as an error that researchers have been trying hard to eliminate. This mindset leads us to reduce or under-exploit the generative potential of generative models. When we switch the objective from prediction to novelty generation, because novelty generation is important in its own right, then it becomes a legitimate research question to understand the full generative potential of these models.

A legitimate question one might ask is the following: "Do we really need a new foundation for studying novelty generation, other than statistical learning theory?"

(a) Both images are generated from a model trained on animals, from Goodfellow (2016). Considering that the model has been trained to generate images of animals (such as dogs), is the left image novel? Is the image at the right noise? Or is it a bad dog? or a new dog? or a new species?



(b) Both sets of images are generated from a model trained on a dataset of digits. Considering that the model has been trained to generate digits, are the images at the left (from Salimans et al. (2016)) novel? Are the images at the right (from Kazakçı et al. (2016)) noise? or bad digits? or new digits? or new types of symbols?

After all, as we acknowledged, this is a basis that proved to be extremely productive. Conceptually, however, it is easy to spot a contradiction between this theory and the task of generating genuine novelty. Within the narrow frame of statistical learning theory, the general goal for a generative modeling is to train a model so that it would be able to reproduce a set of *known* objects with high probability, using objective functions such as *maximum likelihood*. Under such criteria, a genuine novelty (the kind of which was never before seen) is by definition *unlikely*. What is more surprising, despite the restrictive theoretical frame of statistical learning theory, current implementations of generative models *are* generating novelty, as seen in 1.1a and 1.1b and more generally in the literature; see e.g.(Salimans et al., 2016) or (Bengio et al., 2013b)). Currently, this is seen a glitch or a bug: the algorithms developed to respect and implement maximum likelihood and similar frameworks *fail* to respect it.

These elements give enough reason to take this question seriously and to investigate both new theoretical frameworks and the full generative capacity of generative models. However, as we shall see, developing a full-fledged theory of novelty generation within machine learning is not an easy task for several reasons.

For one, generation of novelty is easy. A pseudo-random number generator generates novelty *all the time*. Similarly, a simple image generator that assigns pixel intensities completely randomly will always generate new images. In principle, such a simple generator can generate all Van Gogh and Picasso paintings, as well as new styles of paintings to be invented in the future. But the probability of doing so is so tiny that it will *never* occur in practice. The vast majority of "novel" images that will be thus generated will have no structure at all. In machine learning, it is customary to call such images "*noise*".

What is the difference between generating Van Gogh paintings and generating noise in terms of novelty generation? Originally, noise is a term designating unexplained variance, referring implicitly to the inability of some observer to find recognizable patterns or regularities in the data (e.g., an image would be considered as noise in the sense that it will not be recognizable by a human). Intuitively, separating those two cases requires a selection mechanism, a notion of *value*, to be able to distinguish between useful novelty and unuseful novelty. Although value functions (e.g., utility) over a set of known objects is common place in economics and decision science literatures, what would be a value function for objects that have not yet been generated and are *unknown* is an unanswered question (Kazakçı, 2014).

While the current project aims to answer these questions within the framework of machine learning and generative models, other researchers have sought answers within other literatures and paradigms. For instance, the field of computational creativity (CC) is an inter-disciplinary field whose goal is to study means of gen-

erating new objects computationally. While there exist some work using machine learning models, bulk of the work in CC has exploited evolutionary models for generating novelty. By contrast to the current project, there is much less emphasis in CC to understand foundational elements of novelty generation. CC community seems rather more interested in the end result of the generative process; that is, the artistic or aesthetic value of the generated objects or how computational tools can enhance human creativity. Another fundamental limitation of works in CC, regarding the objectives of this thesis, is the value function (called, fitness function in evolutionary models) that is imposed to the creative agent by the system designer. Since these value functions are fixed beforehand by the programmer of the system, the generated objects reflect her choices rather than the preferences of the machine. While this should not be a problem *if* the value function is known beforehand (and this is rarely the case in a creative process; Hatchuel & Weil (2009)), these systems have little interest in studying how an artificial agent might develop its own preferences during its exploration. Elsewhere (Kazakçı et al., 2016), we called this issue the fitness function barrier. More generally, CC literature does not seek to answer the noise-novelty dilemma that is considered to be fundamental in this research.

Beyond the noise-novelty dilemma, another fundamental question for novelty generation is the relationship between novelty with knowledge about *what already exists*. A large majority of research on computational creativity (e.g., that based on evolutionary algorithms) typically lack a model of knowledge. Better and better representations are obtained at each cycle of the evolutionary computation (according to the predetermined fitness function) but no model of the domain is obtained. More importantly, it is not a general model of the domain that guides the generation. These generate-and-test models have been criticized in design theory (Hatchuel & Weil, 2009), on the grounds that generation and test are not necessarily separate steps, and both are impacted by the knowledge of the agent.

We consider that knowledge is indeed a fundamental aspect of novelty generation process. This makes machine learning models more interesting for the research on the topic. Machine learning algorithms produce data-driven representations of the world. These are, in essence, models of knowledge. The difficulty is that *representations* are not neutral and independent entities, existing on their own. It is very well known in machine learning that, depending on the learning algorithm used and the sample on which they are trained, very different representations can be learned. In fact, representation learning is hard and is one of the major subfields of machine learning. What is less well known is the effect that these different representations might have on what the models can generate as novelty. Indeed, if we reconsider the example of image generation, what we can generate depends significantly on whether we work on raw pixel intensities or on image patches (e.g. 3×3 pixel images). While finding a *good* representation for prediction has been a central topic

in ML, what would be a good representation *for novelty generation* is a question that has not been investigated so far.

Revisiting and better understanding the intricate notions such as noise, value, representation, and knowledge in the context of novelty generation is an ambitious research program. What an alternative framework which would go beyond statistical learning and which would feed research on the topic for generations to come would be is a fundamental yet hard question to answer. Notwithstanding the difficulty, progress must be made. Following the program started in Kazakçı et al. (2016), this project set out to clarify the task ahead, explain limitations of the current body of work, highlight essential problems for novelty generation within machine learning, propose analyzes and potential solutions (e.g., new value functions), and run experiments to study and validate the propositions.

In Chapter 2, we will introduce machine learning, the tasks of supervised and unsupervised learning with some critical review from the perspective of novelty generation. We then present representation learning and its relationship with deep learning and deep generative models. Throughout the chapter, we will suggest that representation learning is task specific, and while existing works involve learning representations that make the task of prediction easier, nothing prevent us to learn representations that are good for generation. In Chapter 3, we will review the literature of two fields that studied novelty generation, namely computational creativity and design theory, and position our work with respect to their core propositions.

Chapter 4 and Chapter 5 are the central chapters of the thesis where we offer conceptual clarifications of the problematic notions we raised during this introduction. In Chapter 4, through a series of simple examples and experiments, we demonstrate the effect of representations on the generation of objects. We demonstrate that a *good* representation can prevent the generation of noise. Moreover, we show that learning a compressed representation can be harmful for novelty generation. Then, through a literature review, we show that the current paradigm of training generative models encourages models that compress the representation. This means that, in theory, they are designed to *kill* novelty generation.

In Chapter 5, we address the problem of the choice of the value function and clarify the relationship between value functions, knowledge, and representation. We present a framework, called KRV, and we propose several categories of value functions, that can potentially form alternatives to maximum likelihood or other related value functions.

In Chapter 6, we explain the benefits of using representations based on deep learning, which we use in the experimental part of the thesis. What we emphasize is that the *mutually constraining effects* of distributed and hierarchical representations of deep nets act as filters and prevent the generation of unrecognizable objects. The representations produced by deep generative models are value-preserving, as

features that were found to be useful to describe the learned domain, remain useful for generating new objects.

Chapters 7 through 9 correspond to our published materials and present a series of experiments investigating various issues we raised regarding the generative capacity of deep neural nets. In Chapter 7, we demonstrate by using a neural network trained on handwritten digits that it is possible to generate symbols that consist in new combinations of *strokes* that do not correspond to known digit categories. More importantly, the experiments demonstrate that even when using a standard objective used in machine learning such as the reconstruction error, it is possible to find models where generation of novelty is possible. This chapter suggests that although the objective function used in generative modeling is to make objects from the data distribution likely and all the other objects unlikely, in practice the objective function is not optimized perfectly, and as a result some objects are generated even if they are unlikely under the (assumed) data distribution, such as combinations of strokes which would not be recognized as digits. We use the experiment to further study the kind of objects that can be generated and note that some clusters naturally form, grouping generated objects that are similar. We call these groups new types (a subset of those are shown in Figure 1.1b, the grid of images at the right).

In Chapter 8, having observed that a model could generate novelty despite the use of an objective function which is not compatible with novelty generation, we propose to go further and find models of that kind automatically. We propose a setup where we evaluate novelty by holding out a set of classes, and training the model on the other (training) classes. We then search for models that generate images that are recognized as images from the held out classes, unseen by the model at training time. More concretely, we train models on digits and find models that generate letters, despite having never seen letters. In order to find those models, we propose a set of metrics to score the models, e.g., *objectness* and *count* metrics. We use those metrics to select models that generate a high number of letters, which we use as a proxy for scoring the models that can generate new combinations of the features learned on digits. Importantly, we show that we can find models that generate digits only, as well as others that can generate new combinations of the strokes. This suggests that there is a difference between the models that generate digits and the models that generate new combinations of strokes, which we shall study in Chapter 10. While the setup has limits, since the metrics are not universal and rely on a specific set of classes (letters), it allows us to analyze and to compare models and it is also an example of an externally imposed value function (defined in Chapter 4).

In Chapter 9, we demonstrate the usefulness of novelty generation on a scientific field, where the goal is to generate molecules from scratch for drug de-

sign. In the classical literature of drug design, most molecule structure generation techniques rely on a library of fragments, which are subgraphs of the molecular graph. The fragments are then combined in chemically meaningful way to form new molecules, using a set of hand-crafted rules that can result into a limited or biased search space. In this chapter, we investigate the use of deep learning models to generate new molecules based on a set of existing drug-like molecules, thus rules are learned automatically from data. We use a domain specific evaluation function to find models that generate drug-like molecules, and we find models that can generate molecules which are better than the existing ones in the training set according to the evaluation function.

In Chapter 10, we make an attempt to explain the difference between models that could generate known objects only (digits) and models that could generate novelty. We show that regularization tools used in the literature, such as sparsity, restriction of the size of the bottleneck, and noise corruption, can reduce the capacity of models to generate novelty. In other words, we show that these regularization tools make the generation of known objects (digits) more likely, making them better models for the current paradigm of generative modeling, but worse for generating novelty. Finally, in Chapter 11, we conclude the thesis and discuss perspectives and future work.

Part I

Literature review and limitations of current approaches

Chapter 2

Machine learning and data-driven models of knowledge

This chapter is an introduction to machine learning and deep learning with a particular emphasis on generative models. In Section 2.1.2 and Section 2.1.3, we will introduce machine learning in general with descriptions of the supervised and unsupervised learning tasks. In Section 2.1.4, we will introduce representation learning. Then, in Section 2.2, we will introduce deep learning and its most used architectures such as the fully-connected neural network, the convolutional neural network (CNN), deep autoencoders, and the recurrent neural network (RNN). Then, we will review different lines of works which studied and experimented with deep generative models. More specifically, we will review works on Generative Adversarial Networks (GANs), others that visualize trained neural networks by optimizing the input to activate some hidden units, neural style transfer, and finally works that use auto-regressive modeling to model a complex probability distribution.

2.1 Machine learning

The goal of machine learning is to automatically learn programs from data. As opposed to hand-written programs, programs that are learned from data can be helpful in situations where the task is hard to write down formally and thus an explicit algorithm solving the task cannot be written down easily (Domingos, 2012).

Consider for instance a classical computer vision problem of image classification. In image classification, the goal is to classify an image into a set of pre-defined classes, like cats and dogs. It is hard to write down mathematically what the concepts of *cat* and *dog* entail. An easier way to specify those concepts would be to give a set of examples of cats and a set of examples of dogs, and learn the cat and

dog concepts from the images directly. This type of learning is called supervised learning, since we are given a dataset of examples with their correct *label*. A label is a semantic information that specify for each example whether it is a dog or a cat. Note that this fundamental setup puts completely aside the question "what is a dog? (or cat)" as it assumes that a trustworthy oracle is able to tell. The goal of the learning task is to use the training dataset to learn a function f which takes as input an example (e.g., an image) and predicts the right label (e.g., dog or cat). Machine learning literature offers a large variety of algorithms to build such an f from labeled data.

Predicting correctly the labels of the training data is trivial - and as of little interest, since it can simply be done by just memorizing the training data into a lookup table. The real value in building a predictive model is to build f in such a way that it generalizes to examples it has never seen - but which corresponds to the classes already learned. Importantly, in supervised learning, as we assume that an oracle is able to tell us the right label corresponding to the examples, we consider that the characteristics of the classes are already known, and the goal is transfer this knowledge from the oracle (e.g., a human) to the machine.

In addition to supervised learning, another task considered in machine learning is *unsupervised learning*. Unsupervised learning (Ghahramani, 2004) is generally defined as the task of automatically discover structure in data, without need of labels, contrary to supervised learning. It is generally accepted that this is an ill-defined task since, in the absence of semantic information, it is hard to determine the quality of the result, whereas it is straightforward in the case of supervised learning, as we can just check whether the predicted output is the same than the true output.

Unsupervised learning can itself be categorized into several tasks, among those tasks we have clustering and dimensionality reduction. The goal of clustering is to discover meaningful *clusters* in the data. Here, *meaningful* is usually considered the equivalent of "interpretable by humans". In dimensionality reduction the goal is to reduce the dimensionality of the data either to visualize it or to extract relevant *features* in order make the task of a supervised learner easier (as it will deal with a smaller input dimensionality), or to do data compression.

An important point to always keep in mind when dealing with machine learning is that modeling data is essentially an optimization problem. This means that what the model is actually achieving depends very much on the objective function given to the optimization problem, as much as the available data.

2.1.1 Discriminative models and generative models

Machine learning literature proposes a variety of models that can be used for supervised and for unsupervised tasks. We usually contrast two kinds of models, the generative models and the discriminative ones.

Discriminative models are usually parametrized functions that can approximate a mapping from inputs to outputs. In the context of supervised learning, the parameters of the discriminative models are optimized to minimize the prediction error, for instance. There are different kinds of discriminative models, and they usually differ in terms of the assumptions they make on the mapping from the inputs to the outputs, and the optimization algorithm which is used to optimize them. In practice, different models are tested and at some point one model among the models which are tested have to be chosen, based on prediction error. It is possible that different models give about the same prediction error, yet be very different in how much they process the data to do the prediction. This suggests there is an underspecification of the models due to how they are trained and evaluated, and nothing prevent us to use the same models for objectives different than prediction.

Generative models are models that can generate data. For instance, common pseudo-random number generators (Devroye, 1986) of uniform or Gaussian distributions can be considered as generative models. In the context of machine learning, generative models are often parametrized probability distributions that we can fit to the data for different purposes. It is usually assumed in machine learning that the data are sampled from a true but unknown probability distribution. In the context of unsupervised learning, generative models can be used to model the data, by minimizing a distance such as Kullback-Divergence (which we define in Section 2.1.3) between the assumed true distribution and the distribution of the model. For instance, a common type of models that are used in unsupervised learning are called the latent variable models (which we describe in Section 2.1.3), and they are often generative models. Latent variables are variables which summarize the input into few bits of information, representing the essence of it. Different latent variable models have different assumptions about how the input is constructed from the latent variables. The procedure of generation of latent variable models consists in first generating the latent variables, which usually have a much smaller dimensionality than the input, and then generating the input by transforming the latent variables. Generative models are often used in unsupervised settings, but can also be used in supervised ones. For instance, in classification, generative models can be used to model the probability distribution of the data for each class/category. In that case, one probabilistic generative model is learned for each class/category, and given an input, the prediction is the class for which the corresponding generative model gives the most probability to the input. Importantly, in the current litera-

ture, even when a generative model is learned in an unsupervised context (without labels), it is ultimately used to enhance or make prediction easier in a supervised task.

In the following, we will define more formally the tasks of supervised and unsupervised learning through a common probabilistic framework.

2.1.2 Supervised learning

In supervised learning, we are given a training set $D_{\text{train}} = \{(x_i, y_i\}, i = 1, \ldots, N_{\text{train}}\}$ where $x_i \in \mathbb{R}^m$ and $y_i \in \mathbb{R}^p$ and a loss function $\mathcal{L}(\hat{y}, y)$ to optimize. The training examples are assumed to be sampled *i.i.d*[1] from an unknown but fixed *true* probability distribution $p(x, y)$ of the joint input and output vector, that is, $x_i \sim p(x)$ and $y_i \sim p(y|x_i), \forall i \in 1, \ldots, N_{\text{train}}$. The goal of supervised learning is to use the training set D_{train} to learn a function $f : \mathbb{R}^m \to \mathbb{R}^p$ which minimize the loss $\mathcal{L}$ over p, that is, we want to minimize

$$\text{EPE} = \int \mathcal{L}(f(x), y)p(x, y)\, d(x, y), \tag{2.1}$$

where EPE is called the *expected prediction error*. EPE measures the amount of generalization error of f over the true data distribution.

In practice, as p is unknown and we only have samples from it, the empirical prediction error is used:

$$\text{Err}(D) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i), y_i), \text{where } (x_i, y_i) \in D \tag{2.2}$$

which estimates EPE through a set of samples D. As we only have samples from p (through the training set) and p is unknown, it is not possible to *optimize* f on EPE directly. f is rather optimized on the training set, and the goal of supervised learning is to find f that minimizes $\text{Err}(D_{\text{train}})$.

Now, given a trained f, it is necessary to *evaluate* its generalization error EPE. Again, as we only have access to samples from p, we cannot evaluate EPE directly. $\text{Err}(D_{\text{train}})$ would be a poor estimator (a very optimistic one) of EPE as f is optimized on the training set. What we usually do is that we split the full training set into two sets: a training set D_{train} and test set D_{test}. We optimize f on the training set by minimizing $\text{Err}(D_{\text{train}})$ and estimate EPE using the test set by evaluating $\text{Err}(D_{\text{test}})$. In practice, a generalization of this validation procedure, called *cross*

[1] Independent and identically distributed. It means that examples that are given to us are sampled independently from the same probability distribution.

validation (Bengio & Grandvalet, 2004) is often used to evaluate the quality of f produced by a learning algorithm.

Two important main tasks in supervised learning are called classification and regression, which differ by the nature of the output y and the loss function $\mathcal{L}$ used. In classification, the output y is a class, a categorical variable, $y \in \{C_1, C_2, \ldots, C_K\}$ where K is the number of *classes*. The loss function usually used is the classification error, $\mathcal{L}(\hat{y}, y) = 1$ if $\hat{y} \neq y$ else 0. In regression, the output y is a real valued vector and the loss function usually used is the squared error $\mathcal{L}(\hat{y}, y) = \|\hat{y} - y\|_2^2$.

2.1.3 Unsupervised learning

The most important change when we move from supervised learning to unsupervised learning is that we no longer have the semantic information provided by the labels. The training dataset becomes thus $D_{\text{train}} = \{x_i, i = 1, \ldots, N_{\text{train}}\}$ where $x_i \in \mathbb{R}^m$. An important assumption in unsupervised learning is that the data is sampled *i.i.d* from an unknown but fixed probability distribution $p(x)$, notes as $x_i \sim p(x)$. Said otherwise, it is assumed that the data has been generated by some phenomena that can be modeled by $p(x)$. Since this true distribution is unknown, the goal in unsupervised learning is to approximate $p(x)$ with a surrogate probability density $q(x)$, where $q(x) \approx p(x)$ according to some similarity measure between probability distributions. This formulation is often called density estimation. The traditional similarity measure used to measure similarity between $p(x)$ and $q(x)$ is Kullback–Leibler (KL) divergence:

$$D_{\text{KL}}(p \parallel q) = \int p(x) \log \frac{p(x)}{q(x)} dx \tag{2.3}$$

$D_{\text{KL}}(p \parallel q)$ has an interesting information theoretic interpretation. If we write it in the following way:

$$\begin{aligned} D_{\text{KL}}(p \parallel q) &= \int p(x) \log p(x) dx - \int p(x) \log q(x) dx \\ &= (- \int p(x) \log q(x) dx) - (- \int p(x) \log p(x) dx) \\ &= H(p, q) - H(p) \end{aligned} \tag{2.4}$$

The first term, $H(p, q)$, is called the cross entropy, and the second one, $H(p)$, the entropy[2]. The cross entropy $H(p, q)$ can be interpreted as the average number of bits that are needed to encode samples from the true distribution p if a coding

[2]More precisely, it is called the differential entropy if p is a continuous distribution.

scheme based on q, rather than p is used. Similarly, $H(p)$ is the average number of bits that are needed to encode samples from the true distribution p if an optimal coding scheme is used, that is, if we use a coding scheme based on p.

Thus, taking a data compression perspective, $D_{\text{KL}}(p \parallel q) = H(p, q) - H(p)$ can be seen as the extra number of bits on average that were required to encode samples from p by using q as as a coding scheme rather than using the optimal one, based on p.

Modeling the surrogate distribution q

The surrogate q can be modeled in several ways. One common way to model q is through latent variable models. Generally, in density modeling, x is a high-dimensional object (e.g., an image or a text). However, it is usually assumed that x lies in a low dimensional manifold, which means x can be characterized by fewer dimensions than m (the *apparent* dimensionality of x). Latent variable models assume the existence of a vector of latent variables denoted by h, which are considered to represent the *essence* of x in few bits of information. This problem of discovering the latent variables can be formalized in a probabilistic framework by considering q describing as the joint distribution of x and latent variables h: $q(x, h)$. $q(x)$ can be written as

$$q(x) = \int q(x|h)q(h)dh \tag{2.5}$$

Note that q is often parametrized by a vector of parameters θ, thus we denote it by $q_\theta(x)$. In that case, the task of unsupervised learning can consist in minimizing $D_{\text{KL}}(p \parallel q_\theta)$. However, we cannot compute D_{KL} because p is unknown. Rather, we use training samples from p and consider the empirical D_{KL} on a set of samples D,

$$D_{KL}(D, q_\theta) = -\frac{1}{N}\sum_{i=1}^{N} \log q_\theta(x_i) - \frac{1}{N}\sum_{i=1}^{N} \log p(x_i), \text{where } x_i \in D. \tag{2.6}$$

We would like to minimize $D_{KL}(D_{\text{train}}, q_\theta)$, that is, find the optimal parameters

$$\theta^* = \arg\min_\theta D_{KL}(D_{\text{train}}, q_\theta) \tag{2.7}$$

Fortunately, as we optimize on θ, the second term, which involves p, can be ignored and thus Eq. (2.7) is equivalent to

$$\theta^* = \arg\min_\theta -\frac{1}{N}\sum_{i=1}^{N} \log q_\theta(x_i), \tag{2.8}$$

or equivalently,

$$\theta^* = \arg\max_{\theta} \frac{1}{N} \sum_{i=1}^{N} \log q_\theta(x_i). \tag{2.9}$$

This optimization problem is called maximum likelihood problem. Said in other terms, the surrogate model q should be built by finding the optimal set of parameters θ in such a way that parameters have highest possible likelihood under q given the observed data - in which case, it is assumed that q approximate well the true distribution p.

Note that the task of building a distribution q such that the observed data would be likely under it is a design choice, and nothing prevent us in principle to build q for other objectives. In Chapter 4.1 we will come back to this point to explain that choosing a different objective can help us go from a learning setup to a design setup.

Inference with a generative model

In latent variable models, we are also interested in doing inference. Inference consists in predicting the latent variables h given x, $q(h|x)$. In order to do inference, we use Bayes' rule

$$q(h|x) = \frac{q(x|h)q(h)}{q(x)}. \tag{2.10}$$

We note that several well known models can be framed under this framework (Ghahramani, 2004), including principal component analysis (PCA), K-means, factor analysis (FA), and independent component analysis (ICA).

Depending on the choice and complexity of q, evaluating either $q(x)$ or $q(h|x)$ or both of them can be intractable, that is, difficult to compute exactly. In those cases, one often has to resort to approximations.

2.1.4 Representation learning

In computer science, the term representation refers to a description of data that can be processed by computers. In machine learning, data are usually represented as vectors in a d dimension space (e.g., points in $\mathbb{R}^d$). One of the specificities of machine learning is that the input dimensionality is usually very big. For instance, for images, there is one (grayscale) or three (color) dimensions for each pixel, which implies that the size of the input representation can grow very rapidly with the size the image[3]. A well known issue in machine learning is the so-called *curse of di-*

[3]For instance, a colored image of size 256×256 would be represented as a vector of approximately 200K dimensions.

mensionality (Bellman, 1957). The curse of dimensionality occurs with inputs of high dimensionality, and implies that as the number of dimensions d of the input increase, the number of combinations that the input vector can take grows exponentially, making generalization very difficult in the case of supervised learning, because it would require a huge number of training examples to be able to predict correctly unseen examples. Fortunately, real data that machine learning practitioners deal with, even when it is very high dimensional, usually does not cover the full exponential space of combinations in the input space. Rather, it is observed that the real life data tend to concentrate into a low dimensional non-linear manifold (Kégl, 2003), with a dimensionality which is much lower than d.

In practice, to circumvent this problem, machine learning practitioners usually re-represent the data based on their knowledge of the domain. This step called *feature engineering*, and is one of the most difficult steps in machine learning pipelines. Feature engineering consists of manually designing features (representation of data) for a given task (e.g., classification) to alleviate the problem of finding a good model. As an example, if the goal is to recognize images of handwritten digits, representing images by the location and type of strokes (curvy, straight, etc.) instead of raw pixel intensities will make the task of machine learning much easier because raw pixel values, if taken individually, convey little information about a high level concept like 'being digit 5'.

Feature engineering is usually very costly and with a lots of trial and error. An alternative to feature engineering is to *learn* representations from data automatically. Learning a good representation of the data is so important in machine learning that it is considered as an entire subfield of machine learning and it is called *representation learning*. One of the main assumptions which is made in representation learning algorithms is that the data belongs to a low dimensional manifold m, much lower than the *apparent* dimensionality of the data d, that is, $m \ll d$. A representation is considered good if it helps the learning task. Representation learning algorithms are usually used for prediction tasks, but also for density estimation tasks. However, even in the case of density estimation, which is usually done in an unsupervised context (without labels), the ultimate goal is often to use the learned representation for prediction. Importantly, in the context of prediction, different learned representations can in principle lead to about the same predictive error, suggesting that the representation is underspecified (different representations can lead to about the same prediction error). It would be tempting to learn then exploit the representations for a different task, such as novelty generation which we discuss in Section 4.1. In the following, we will review deep learning, an important family of representation learning algorithms that have shown important empirical successes recently.

2.2 Deep learning and generative models

An important family of representation learning algorithms are called *deep learning*. Deep learning (LeCun et al., 2015) is a machine learning paradigm which had major impact in several fields like computer vision (Krizhevsky et al., 2012), speech recognition (Dahl et al., 2012), and natural language processing (Collobert et al., 2011). One of the main characteristics of deep learning is that it can automatically learn features from raw data (e.g., from pixels or audio signals), avoiding the need for feature engineering, with the cost of having to tune its architecture and numerous parameters.

The term *deep learning* mainly refers to the idea of using multiple levels of representations with growing abstraction. This idea is implemented through *neural networks*, which are a family of functions mapping an input space to an output space through a set of non-linear transformations, called "layers", which can be seen as intermediate representations of the data.

There are many kinds of layers, like for instance fully-connected and convolutional layers. Layers differ by how they transform their input to an output. These different kinds of layers are composed together, in a directed acyclic graph (DAG), which we call the *computational graph*. Each kind of layer has potentially learnable parameters called *weights*, which are adjusted in the training phase to optimize the loss function. A layer has also *hyperparameters* which are parameters that have to be fixed before the training phase (e.g., number of units). The way these layers are chosen (either by the practitioner or in an automatic way) to be composed together and the choice of their hyperparameters determine the *architecture* of a neural network.

One particularity of neural networks is that they can be trained in an *end-to-end* manner. The concept of *end-to-end* means that if we have a pipeline with multiple steps, we have the possibility to adjust the parameters of all the steps together to optimize the final objective, rather than optimizing each step separately in a greedy way. In neural networks, layers can be seen as intermediate steps of a pipeline, going from the input to the output. End-to-end training in neural networks means that all the parameters of all the layers, starting from the input to the output, can be adjusted to optimize the loss function, which usually operates on the output space (e.g., we want the predicted output to be close the desired output).

End-to-end training is possible because layers are chosen to be differentiable functions. This means that it is possible to use stochastic gradient descent (SGD) or variants of it to optimize the loss function, by computing the gradients of the parameters of all the layers together with *backpropagation* (Rumelhart et al., 1986). Backpropagation is a dynamic programming algorithm, which consists in a recursive application of the chain rule starting from the outputs and going backwards

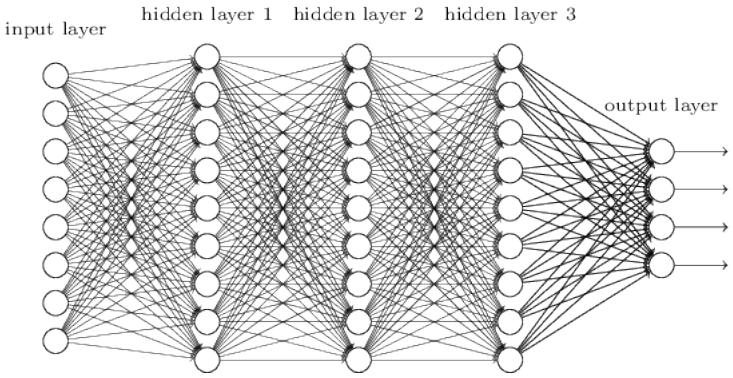


Figure 2.1: A visualization of fully-connected neural networks. See the text for more details. This picture was taken from Nielsen (2015).

through layers to compute the gradients of all the weights. The gradients are then used to update the weights in order to optimize the loss function.

2.2.1 Fully-connected neural networks

Fully-connected neural networks, also called multi-layer perceptrons (MLP), are the most basic neural network architecture. They map an input vector $x \in \mathbb{R}^m$ into an output vector $y \in \mathbb{R}^p$ through a series of linear transformations followed each by a non-linear function applied element-wise. A fully-connected layer maps from $\mathbb{R}^{n_{\ell-1}}$ to $\mathbb{R}^{n_\ell}$ where n_ℓ is the dimensionality of the vector in the l-th layer, and it is parametrized by a *weight* matrix W_ℓ with $n_{\ell-1}$ rows and n_ℓ columns, a bias vector b_ℓ with n_ℓ components, and a non-linear function called the *activation* function g_ℓ, which is applied element-wise. Thus, a fully-connected layer consists in a function $f_\ell : \mathbb{R}^{n_{\ell-1}} \to \mathbb{R}^{n_\ell}$, where $f_\ell(x) = g_\ell(W_\ell x + b)$. A fully-connected neural network f is a composition of fully-connected layers, that is, $f = f_L \circ f_{L-1} \circ \ldots \circ f_2 \circ f_1$ where L is the number of layers, or the *depth* of the neural network, and $n_0 = m$, $n_L = p$. The components of the vector h_ℓ obtained after applying $h_\ell = (f_\ell \circ f_{\ell-1} \ldots \circ f_2 \circ f_1)(x)$ to an input are called the hidden layer activations or hidden units or neurons of the layer ℓ in the literature. The "neurons" of h_ℓ are said to be connected to the "neurons" of $h_{\ell-1}$ through the weight matrix W, that is, the j-th hidden unit of ℓ, $h_{\ell,j}$ is said to be connected to the hidden unit i of $\ell - 1$, $h_{\ell-1,i}$ through the weight $W_{i,j}$. In Figure 2.1, we can see a visualization of a fully-connected neural network. Each circle corresponds to a unit of a layer, and the edges represent the weights (connections) between units.

2.2.2 Convolutional neural networks

Convolutional neural networks (CNNs) (LeCun et al., 1998) are a special kind of neural networks which can be seen as a constrained version of fully-connected neural networks. Convolutional neural networks have advanced the state of the art of computer vision since 2012 in several vision tasks, including image classification (Krizhevsky et al., 2012), detection (Ren et al., 2015), segmentation (Long et al., 2015), retrieval (Sharif Razavian et al., 2014), pose estimation (Toshev & Szegedy, 2014), captioning (Xu et al., 2015), face recognition (Taigman et al., 2014; Schroff et al., 2015), and super resolution (Dong et al., 2014; Ledig et al., 2016). Although their application is mainly in computer vision, they have been successfully applied as well to speech recognition (Abdel-Hamid et al., 2014), and even in natural language processing (NLP) (Kim, 2014; Zhang et al., 2015; Kalchbrenner et al., 2016; Gehring et al., 2017).

In fully-connected neural networks, layers are organized into a vector of units while in convolutional networks they are organized into tensors. When dealing with images, layers are tensors of *order* 3, also called *volumes*, where each tensor is indexed by the width, height, and depth. Convolutional neural networks are designed to take into account the fact that the inputs are arranged spatially in a grid of values, contrary to fully-connected networks which just see an unordered and flattened vector of values.

There are two main key ideas in the design of CNNs that make it powerful. The first is *sparse connectivity*, units in a given layer are only connected to a subset of units in the previous layer, contrary to fully-connected neural networks, where the connections are said to be *dense*. The second is *weight sharing*, which means that different units in a layer re-use the same weights than other units. Below we will explain how convolutional neural networks are constructed, and explain how they implemented these two ideas.

Convolutional neural networks are characterized by two main layers, convolutional layers and pooling layers. Each convolutional layer performs a discrete convolution operation on its input using a set of d_ℓ *filters*, which are arranged in a 4-th order tensor F_ℓ indexed by $(w_\ell^F, h_\ell^F, \mathrm{d}_{\ell-1}, \mathrm{d}_\ell)$, where w_ℓ^F is the width of the filters, h_ℓ^F the height of filters, $\mathrm{d}_{\ell-1}$ the input volume depth, and d_ℓ the number of filters which also determine the output volume depth.

Convolutional layers convert an input volume of shape $(w_{\ell-1}, h_{\ell-1}, d_{\ell-1})$ to an output volume of shape (w_ℓ, h_ℓ, d_ℓ) where w_ℓ, h_ℓ, and d_ℓ are respectively the width, height, and depth of layer ℓ. Each slice of the input or the output volumes on the depth component is called a *feature map*, thus a feature map of the layer ℓ is a grid of shape (w_ℓ, h_ℓ) and the layer ℓ have d_ℓ feature maps. When we have colored images, the input volume of the first layer is a volume of shape $(w_0, h_0, 3)$ where

w_0 is the width of the image, h_0 the height of the image, and 3 are the number of feature maps, which correspond to color channels (red, green, and blue). To obtain the j-th feature map of the output volume, a 3D discrete convolution is applied between the input volume, which has a shape of $(w_{\ell-1}, h_{\ell-1}, d_{\ell-1})$, and the j-th filter of F_ℓ, which is a volume of shape $(w_\ell^F, h_\ell^F, d_{\ell-1})$. When a *valid* (Dumoulin & Visin, 2016) discrete convolution is applied, the output volume has a smaller size than the input volume, where $w_\ell = w_{\ell-1} - w_\ell^F + 1$ and $h_\ell = h_{\ell-1} - h_\ell^F + 1$.

The discrete convolution operation is used extensively in image processing to perform various operations like edge detection, blurring, etc. The same concept is used in convolutional layers but the filters F are not pre-defined and fixed like in image processing, they are rather adjusted and learned from data. Also, convolutions are useful because they allow weight sharing (the same filter is applied everywhere), this reduces the number of parameters by a big amount and make the number of weights (filters) independent of the size of the input, compared to fully-connected layers where the number of weights is proportional to the size of its input. Additionally, as the filter is applied everywhere, we are expecting that the filter will be useful in different locations, which makes sense if an object that the filter detects (for instance it makes sense for an edge detector) can appear everywhere in the image. The convolution operation also implements sparse connectivity because the value of output units are only connected to a subset of input units as the convolution operation is local. Like fully-connected layers, convolutional layers are also followed by a non-linear activation function which is applied element-wise.

The purpose of pooling layers is to reduce the size of the input by a certain ratio by summarizing blocks of typically size 2×2 of the image using the mean or the max operation. However, this size reduction comes with a price: we lose some information about the exact position of the detected features. But it turns out that this can be helpful in the context of prediction, first it helps to reduce memory and computational power required in subsequent layers, and losing the exact position of the features can give some translation invariance (Scherer et al., 2010). Pooling layers do not have learnable parameters, they only have hyperparameters, which are the strides and the summary operation. The strides (one for width, one for height) define the ratio by which we subsample, if it is 2 for both width and height dimensions we end up dividing the size of the input width and height by 2. The most widely used summary operation is the *max* because it works well in practice, in that case we call the layer the *max-pooling* layer.

Figure 2.2 shows a typical convolutional neural network, which starts by the input image and consists in a series of two blocks of convolutional layers followed by a pooling layer. $\text{ReLU}(x) = \max(x, 0)$ refers to the non-linear activation function commonly used in convolutional neural networks. Each column refers to a layer, and within each layer, the squares refer to the feature maps, each feature

map can be seen as a pattern detector, where upper layers can be considered as more abstract pattern detectors than lower ones in the sense of detecting patterns on a larger receptive field than lower layers, where the receptive field is the grid of the pixels on the input image that affect the feature map units on a given layer. The series of convolutional and pooling layers are usually followed by a series of fully-connected layers, denoted in the picture as *FC*, for instance to predict a vector of probabilities for each class in a classification task, like in the figure.

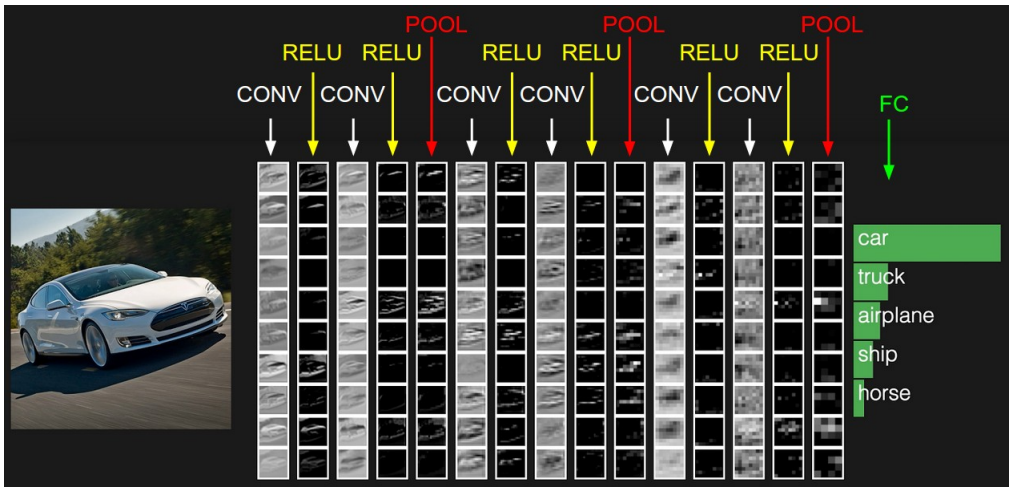


Figure 2.2: A visualization of a convolutional neural network. See the text for more details. This picture was taken from https://github.com/cs231n/cs231n.github.io.

2.2.3 Autoencoders

Autoencoders are a special kind of neural networks which are used to do representation learning when labels are not available, that is, they are a way to learn a data representation in an unsupervised way. Autoencoders consist in two neural network modules, an *encoder* $h = \text{enc}(x)$ and a *decoder* $\hat{x} = \text{dec}(h)$. The encoder *encodes* the input x into a representation h, which we call the *code* or hidden representation, while the decoder reconstructs the input $\hat{x}$ from the code h. The parameters of the autoencoder are optimized in a way that the reconstruction error is minimal, for instance using squared error $\|x - \hat{x}\|_2^2$.

The code is usually restricted in some way so that it can compress the input in few bits, otherwise, if there is too much capacity in the code (no restrictions), the network learns the identity function and the code learned is not useful (as a representation). Usually, the code learned by the autoencoder is meant to be useful

when it can improve the performance in a supervised learning task, compared to using the raw representation.

Autoencoders can be interpreted in several ways. If the encoder and decoder are just linear transformations and the reconstruction error is the mean squared error between the reconstructed input and the real input, then it was shown in Baldi & Hornik (1989) that the autoencoder behave like Principal Component Analysis (PCA). However, when the encoder and the decoder are non-linear, autoencoders behave differently than PCA and generalize it. Performing an autoencoder can be seen like projecting the input into a curved manifold, whereas performing PCA is projecting the input into a linear manifold.

Autoencoders have also a vector field interpretation (Kamyshanska & Memisevic, 2015). Consider the reconstruction function of the autoencoder: $r(x) = \text{dec}(\text{enc}(x))$. Autoencoders can be seen as defining a vector field which assigns, for each point x, a vector $r(x) - x$. The vector field defined by the autoencoder can be seen as a set of paths which point towards regions of high probability density of the data distribution. Thus, following the vector field can be thought as doing gradient ascent in the scalar field, which represents the unnormalized probability density, or the score that assigns how much the autoencoder "likes" a point.

In the literature, different types of autoencoders have been proposed, and they mainly differ on the mechanism which is used to restrict its capacity to force it to learn a useful representation. At this point we leave "usefulness" open, but one way to see these variants is that they all want to avoid learning the trivial identity function. The most basic way to restrict the capacity of the representation is to use a bottleneck layer which has a much smaller dimensionality than the input. In the case of a deep autoencoder the architecture of the neural network is usually designed as a pyramidal structure with the bottleneck layer in the middle, see Figure 2.3 for an illustration.

One well-known and successful variant of the autoencoders is denoising autoencoders (Vincent et al., 2010). The idea of denoising autoencoders is to force the network to learn a robust representation by learning to reconstruct the input using a corrupted version of it. More concretely, say x' is a corrupted version of the input (e.g., x' is x with adding some random noise or zeroing out randomly some units), we will force the autoencoder to *denoise* its input by minimizing $\|x - \text{dec}(\text{enc}(x'))\|_2^2$ instead of $\|x - \text{dec}(\text{enc}(x))\|_2^2$.

Denoising autoencoders have a probabilistic interpretation and have been shown to be generative models which approximate the assumed true probability distribution p. Bengio et al. (2013b) show that training the autoencoder to reconstruct its corrupted input leads to approximating a Markov chain with two operators, a corruption operator $C(x'|x)$ and a reconstruction operator $p(x|\hat{x})$ where $\hat{x} = \text{dec}(\text{enc}(x'))$ and the stationary distribution of the ideal Markov chain we

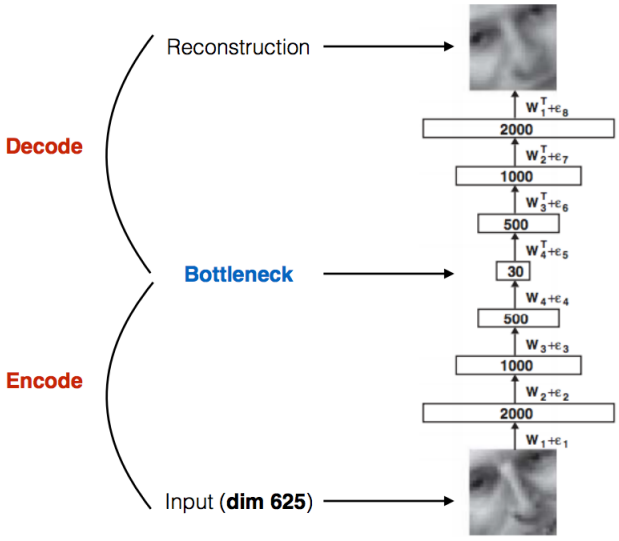


Figure 2.3: A visualization of an autoencoder with a bottleneck from (Hinton & Salakhutdinov, 2006).

approximate is the true distribution, meaning that if we repeatedly apply the two operators of the Markov chain for a large number of steps we obtain samples from the true distribution.

Another way to constrain the representation of autoencoders is to use the contractive criterion (Rifai et al., 2011). The contractive criterion encourages the hidden representation to be insensitive to the input by penalizing the gradients of the representation with respect to the input $\sum_{i,j}(\frac{\partial h_i}{\partial x_j})^2$ where h_i is the i-th hidden unit and x_j is the j-th input unit of a training example. When optimizing the penalization alone, we obtain a constant representation that throws away all the information about the input and thus cannot distinguish between different examples. On the other hand, when optimizing the reconstruction objective alone, the network is forced to have a representation that keeps information about training examples, so that it can reconstruct them. The result of using the reconstruction error along with the penalization is that the network only keeps the information that is necessary to reconstruct the training examples, and throws away the rest. In other words, it discovers a hidden representation which encodes useful information about training data and can distinguish between training examples, because a hidden representation that varies for different inputs but does not encode useful information about the training data would cause a bad reconstruction error and a high value for the penalization. As a result, when thinking from the manifold learning perspective, if we

assume that most data lies in low dimensional manifold, contractive autoencoders can be seen as a way to make the hidden representation changing with respect to important variations corresponding to the manifold (e.g, rotation or translation of the images) and invariant to all the variations which are orthogonal to the manifold. For instance, if we train a contractive autoencoder on a dataset containing the digit 8 where the only variation is translation, rotating an image of the digit 8 would not change the representation because it is a transformation that do not exist in the training set, thus not allowed. The representation will only vary if we translate an image of the digit 8.

Another way to constrain the representation of autoencoders is to use sparsity. The idea of sparse autoencoders (Ng, 2011) comes from *sparse coding* (Olshausen & Field, 1997; Lee et al., 2007; Mairal et al., 2009). Sparse coding consists in finding a linear decomposition of a signal (e.g., an image) into a limited number (sparse) set of primitives (e.g., oriented edges), where the total number of primitives is typically greater than the dimension of the signal (contrary to PCA), forming an *overcomplete* basis. In other words, in sparse coding, each signal x is represented as a weighted linear combination of atoms: $x = \sum_{j=1}^{k} \mathcal{D}_j \alpha_j$ where $x \in \mathbb{R}^m$, $D_j \in \mathbb{R}^m$, $\alpha_j \in \mathbb{R}$ and most α_j are zero. The sparse coding objective can be formulated as

$$\sum_{i=1}^{n} \left\| x_i - \sum_{j=1}^{k} \mathcal{D}_j \alpha_{i,j} \right\|_2^2 + \lambda \sum_{i=1}^{n} \|\alpha_i\|_1, \tag{2.11}$$

where $x_i \in \mathbb{R}^m$ is a training example, n the size of training data, $\alpha_{i,j}$ is the coefficient of the j-th atom for the i-th example, and $\mathcal{D}_j \in \mathbb{R}^m$ is the j-th atom. The optimization is performed simultaneously over α and D and the goal is thus to minimize the reconstruction error (first term) and make the coefficients as sparse as possible (second term). λ controls the desired amount of sparsity. This objective is very much similar to what autoencoders do, where the decoder is a linear layer for which the weights are D and the hidden representation of each training example $x_i \in \mathbb{R}^m$ can be considered as $\alpha_i \in \mathbb{R}^k$. The main difference is that in sparse coding there is no encoder, rather, a representation α_i is learned for each training example[4]. In sparse autoencoders, the idea is to optimize a similar objective but in addition to that, we also learn an encoder function that can be applied to new examples after training.

Different ways of training sparse autoencoders have been proposed in the literature. Ng (2011) trains autoencoders with a reconstruction error objective and a regularization objective which constrains the units of the hidden representa-

[4] In the literature, this is called non-parametric learning.

tion to be active only few times on the training set. The hidden units are constrained to be between 0 and 1 using the sigmoid non-linearity. The regularization objective forces the mean of each hidden unit to be close to 0 by minimizing $\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$ for each hidden unit where $\hat{\rho}_j$ is the mean of the j-th hidden unit over the training set and ρ is a constant close to 0 that determines the amount of desired sparsity. This setup can also be seen as interpreting each hidden unit as a Bernoulli random variable which is constrained to have a mean close to zero.

Other ways of obtaining a sparse representation are the recently proposed k-sparse autoencoders (Makhzani & Frey, 2013) and Winner-Take-All (WTA) autoencoders (Makhzani & Frey, 2015). In the k-sparse autoencoders, rather than adding a regularization objective, the sparsity is explicitly done by sorting the hidden units of each example and setting the smallest $k\%$ hidden units of each example to be 0 (to obtain a sparsity rate of $k\%$) in the forward pass, and backpropagating only through the non-zero hidden units in the backward pass. This issue with k-sparse autoencoders is that it can lead to the phenomenon of dead units, which means that it can cause some units to never be backpropagated through and thus never used by any example (their value is always 0). To overcome this issue, (Makhzani & Frey, 2015) proposed the *lifetime* sparsity which rather than setting the smallest hidden units of each *example* to be 0, it sets the smallest $k\%$ activations of each *hidden unit* through a mini-batch of examples to 0. This way, we can achieve exactly the same sparsity rates than the k-sparse autoencoder, but the phenomenon of dead units does not happen because each hidden unit is necessarily used and backpropagated through in each mini-batch of examples. Makhzani & Frey (2015) also proposed *spatial* sparsity which is specially designed for convolutional autoencoders (autoencoders with convolutional layers), for which only the maximum activation of each feature map is kept (the other activations are set to zero).

Variational autoencoders (VAEs) are another way to regularize the autoencoders (Kingma & Welling, 2013), by constraining the latent variables to follow a simple distribution. In variational autoencoders, we consider the latent variables z as stochastic, and we usually constrain them to follow a simple distribution like a spherical Gaussian. The generative process is thought of as first generating the latent variables from their prior, $z \sim p_\theta(z)$, then generating the inputs using a decoder, which is a parametrized density $x \sim p_\theta(x|z)$, where we compute the parameters of the input distribution from z, usually using neural networks. At the same time, a "variational" inference network $q_\Phi(z|x)$ or encoder is trained to approximate $p(z|x)$, using a neural network as well. The loss function used in VAE is called the variational lower bound (ELBO), and it is decomposed into two terms, a reconstruction term and a regularization term. In the reconstruction term,

$\mathbf{E}_{q_\phi(z|x)}[\log p_\theta(x|z)]$, we want to reconstruct the inputs from the latent variables, just like ordinary autoencoders. In the regularization term $D_{\text{KL}}(q_\phi(z|x) \parallel p_\theta(z))$, we want the latent variables $q_\phi(z|x)$ to not be too divergent from the prior $p_\theta(z)$, which is usually chosen as a spherical Gaussian $p_\theta(z) = \mathbb{N}(0, I)$

2.2.4 Recurrent neural networks

Recurrent neural networks (RNNs) are special neural networks that map a variable-length input sequence $x_1, x_2, \ldots, x_T$ to a variable length output sequence $y_1, y_2, \ldots, y_{T'}$, where T is the sequence length of the input and T' the sequence length of the output. Just like convolutional neural networks, RNNs implement the idea of weight sharing and the number of parameters do not grow with the size of input or the output sequence contrary to the fully connected neural networks. However, RNNs are *recurrent* and are contrasted with *feed-forward* neural networks (like fully-connected and convolutional neural networks) because contrary to feed-forward neural networks the computational graph of RNN contain a loop (a node that connects to itself). The most important component of RNNs is their *hidden state* and can be seen as as a summary of all what the RNN has seen up to time t. The hidden state is defined as

$$h_t = f_\theta(h_{t-1}, x_t), \tag{2.12}$$

where h_t, the hidden state at timestep t, is a vector of numerical features and f is a function parametrized by θ that predicts the next hidden state given the previous hidden state and the current input. RNN contains a loop because the hidden state is connected to itself, we can see that directly in Equation 2.12 where h_t is a function of h_{t-1}. RNNs also predict an output for each timestep. The output of the RNN at each timestep is predicted using the hidden state

$$y_t = g_\Phi(h_t) \tag{2.13}$$

where g is a function parametrized by Φ.

A typical use case of RNNs is language modeling, where the goal is to model the probability distribution of a sequence of symbols

$$p(x_1, x_2, \ldots, x_T) = \prod_{j=1}^{T} p(x_j|x_1, \ldots, x_{j-1}), \tag{2.14}$$

where each term $p(x_j|x_1, \ldots, x_{j-1})$ is parametrized by the RNN as $p(x_j = c|x_1, \ldots, x_{j-1}) = g_\Phi(c|h_j)$ and $g_\Phi(c|h_j)$ is the probability of c as predicted by the RNN given the current hidden state. Suppose we are given a dataset of sequences $\{s_1, s_2, \ldots, s_N\}$,

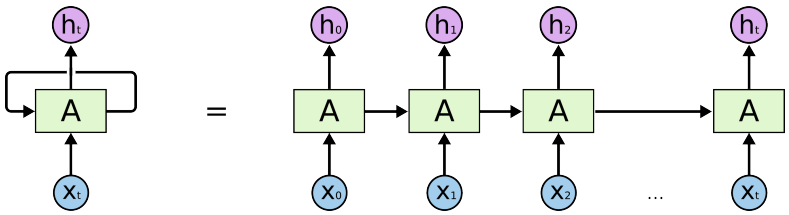


Figure 2.4: Left: computation graph of an RNN where X_t is the input at timestep t, h_t is the hidden state at timestep t, and A the recurrence function that outputs h_t given h_{t-1} and X_t. Right: unfolded version of the computation graph. The unfolded graph can be used to compute the gradients of the parameters with backpropagation, just like fully connected and convolutional neural networks. The figure is taken from (Olah, 2015).

where $s_i = s_{i,1}s_{i,2}\dots s_{i,\text{len}(s_i)}$ and $\text{len}(s_i)$ is the length of s_i and $i = 1,\dots,N$. RNNs are typically trained with maximum likelihood,

$$(\theta^*, \Phi^*) = \arg\max_{\theta,\Phi} \log P(s_1, s_2, \dots, s_N|\theta, \Phi), \tag{2.15}$$

where

$$\begin{aligned} \log P(s_1, s_2, \dots, s_N|\theta, \Phi) &= \sum_{i=1}^{N} \log P(s_i|\theta, \Phi) \\ &= \sum_{i=1}^{N} \sum_{j=1}^{\text{len}(s_i)} \log P(s_{i,j}|s_{i,1}, s_{i,2}, \dots, s_{i,j-1}, \theta, \Phi) \\ &= \sum_{i=1}^{N} \sum_{j=1}^{\text{len}(s_i)} \log\, g_\Phi(s_{i,j}|h_j) \end{aligned} \tag{2.16}$$

and $h_j = f_\theta(h_{j-1}, x_j)$. The RNN parameters (θ, Φ), used respectively in f and g, are optimized to maximize Equation 2.16. The gradients are computed using backpropagation through time (BPTT) (Werbos, 1990), which is an application of backpropogation to computational graphs with recurrence (loops). BPTT is the usual backpropagation applied in the *unfolded* computation graph. The unfolded graph is a graph where the hidden state loop is transformed into a forward computation with a replication of the hidden state update operation as many times as the total number of timesteps. See Figure 2.4 for an illustration.

There are different parametrizations of RNNs, and they differ mainly by how the recurrence function f is parametrized. For instance, "vanilla" RNNs (Karpathy

et al., 2015) use the recurrence function

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b), \tag{2.17}$$

where h_t, the hidden state at timestep t, is a vector of k dimensions, x_t the input at timestep t a vector of m dimensions, W_x is a matrix of shape (m, k), W_h is a matrix of shape (k, k), and b a vector of k dimensions. Training RNNs can be difficult because they suffer from two fundamental issues known as *vanishing gradient* (Bengio et al., 1994; Hochreiter, 1998) and *exploding gradient* (Pascanu et al., 2012) problems. Those problems arise both because of the saturating non-linearity used in the recurrence equation ($tanh$ in Equation 2.17) and the fact that we multiply each timestep by the same weights W_h, and are aggravated with more timesteps. It is easier to see it with scalars. Let us suppose the loss function is L and we have a loss in each timestep L_t. Suppose want to compute the gradients $\frac{\partial L_t}{\partial w}, t = 1 \dots T$, in order to use stochastic gradient descent and we have an unfolded graph with T timesteps. Suppose $h_t = \tanh(wh_{t-1}), t = 1 \dots T$. In the following w and h_t are simply scalars. By using BPTT, we obtain:

$$\frac{\partial L_t}{\partial w} = \sum_{i=1}^{t} \frac{\partial L_t}{\partial h_i} \frac{\partial h_i}{\partial w} \tag{2.18}$$

$$\begin{aligned} \frac{\partial L_t}{\partial h_i} &= \frac{\partial L_t}{\partial h_{i+1}} \frac{\partial h_{i+1}}{\partial h_i} \\ &= \frac{\partial L_t}{\partial h_{i+1}} \tanh'(wh_i)w \\ &= \prod_{j=i}^{t} (\tanh'(wh_j)w) \\ &= w^{t-i+1} \prod_{j=i}^{t} \tanh'(wh_j) \end{aligned} \tag{2.19}$$

We can notice two problems. The first one which we can see in Equation 2.19, is that as we go back over timesteps, we multiply the gradients by the product of the derivative of tanh and w. If their product is smaller than 1 the gradients over time converge exponentially to zero (vanishing gradient), whereas if their product is bigger than 1 the gradients get bigger and bigger and explode (exploding gradient). The derivative of tanh is smaller than 1, so it encourages the vanishing gradient problem. The weights are usually initialized to be a small value between -1 and 1, thus in the beginning of training it is very easy to be in a situation of vanishing

gradients. The second problem, which we can see in Equation 2.18, is that the gradients of w are summed over all previous timesteps, where the timesteps close to t have more importance (or less importance for exploding gradients) than timesteps in the beginning. An implication is that RNNs cannot handle long range dependencies. For instance, memorizing a value in the hidden state for long timesteps would be a difficult task. Exploding gradients are simply solved by making the gradient weight norms smaller if they exceed some threshold (Pascanu et al., 2012). Vanishing gradients are harder to deal with. Long short-Term Memory (LSTM), a parametrization of RNNs proposed by Hochreiter & Schmidhuber (1997), alleviates the vanishing gradient problem by introducing the concept of *gates* which are a way to learn to keep the desired information intact when needed rather than changing the hidden state abruptly each timestep by multiplying by W_h. Also, LSTMs use the identity function as an activation function in the recurrence relation rather than a saturating non-linearity such as tanh, so it avoids the repeated application of its derivative, which is smaller than 1 and can encourage the gradient to vanish.

2.2.5 Deep generative models

Recently we have seen a growing interest in building generative models based on data in the deep learning community, mainly for images, but also on text and music. In the following, we present several lines of works that cover generative models used for image generation. The following is a detailed literature review which can be skipped by the reader.

Activation maximization

For instance, a line of papers from evolving AI lab (http://www.evolvingai.org) proposed several ways to do *activation maximization* based on deep neural net image classifier. Activation maximization consists in optimizing an image in the pixel space in such a way that when that image is fed to a neural network (usually a classifier), it would maximize a chosen unit in an intermediate layer. Although their main goal was to visualize what the *function* of each *neuron* is, this optimization procedure coupled with the neural network can itself be seen as a generative model.

In their first paper about activation maximization, Nguyen et al. (2015a) propose to use evolutionary algorithms to find images that *fool* a deep neural net, that is, where the deep neural net predicts with high confidence a class which is not related to the content of the image. They use direct and indirect encodings, where the direct encoding means that the evolutionary algorithm is applied on the pixel space, whereas in the indirect encoding, they use a certain kind of parameterizable graphics renderer (called CPPN (Stanley, 2007)) and they rather optimize its

parameters. As the renderer has components that have properties like symmetry and repetition, it produces images that are more regular (compared to optimizing on pixel space). The result is that doing optimization directly in the pixel space leads to noise, but for which the deep net can recognize with high confidence for some class, although they found it much difficult to do this optimization for Imagenet compared to MNIST. The optimization was much easier with the graphics renderer, and it produced images with regularities but which mostly were unrelated to the classes, however they did observe in some cases that the images had certain features associated to the class. See Figure 2.5.

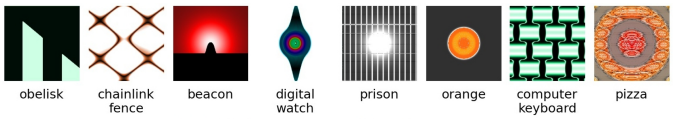


Figure 2.5: "Synthetic" objects from Imagenet categories from Figure 7 of (Nguyen et al., 2015b)

Yosinski et al. (2015), having observed the advantage of using a renderer, acknowledge the importance of having a good image prior when optimizing the images in the pixel space to avoid producing "unnatural" images, as the renderer can be seen as a prior. In that paper, they rather use gradient descent on the pixel space but coupled with a hand-crafted regularization criterion on the pixels to make the images look realistic and interpretable. They obtain indeed better images than when pixel space optimization is used alone, without regularization.

Nguyen et al. (2016c) extend the framework by hypothesizing that "neurons" are multifaceted, that is, the same neuron can recognize different facets/attributes of the same concept. They locate the facets of a given class of images by doing clustering, each facet is hypothesized to form a cluster. Thus, they take cluster centroids and use them as representative for each facet, then do activation maximization by initializing with the centroid. Thus, for the same class/category, they could obtain different facets, each one obtained by initializing from a different centroid, found by clustering.

Although the regularizations improved a lot the *naturalness* of the images, they were hand-crafted (regularizations). Nguyen et al. (2016a) attempted to use a model, learned from data, as an image prior. They obtain much better results (see Figure 2.6). Their prior is an image generator that takes as input a code, and outputs a synthetic image. The code itself is obtained by using a layer from a neural network trained on classification. Thus, given the code, they train the image generator to invert the code. Now, in order to visualize a neuron, they optimize the code so that it outputs an image for which the neuron is activated the most. The image

generator is a strong prior because it is trained to output only realistic images. Interestingly, the image generator is trained with 3 losses at the same time to obtain realistic images, 1) the generated image should be close to the real image in pixel space 2) the features of the generated image should be close to the features of the real image for some chosen feature space 3) like in GANs, a discriminator cannot discriminate between the generated and real image. Another interesting observation was the impact of the layer chosen originally for the code. When using convolutional layers, it produced images with repeatable patterns, like one can see in deep dream (Mordvintsev et al., 2015). However, when using fully-connected layers, they could obtain images with much better global structure. The reason is that because low-level layers has a smaller local receptive field, while fully-connected obtain information from all image parts. Also, in an interesting experiment, they propose to use their framework to generate creative images by activating multiple neurons at the same time. They show that it is possible for instance to generate a hybrid of two classes by activating both classes at the same time. See Figure 2.7 for an example.

Figure 2.6: Images generated from Imagenet classes. The picture is from Nguyen et al. (2016a).

One issue they found is that although the images were realistic, they lacked the intra-class diversity found in the real images, even with different random initializations. The goal of Nguyen et al. (2016b) was to increase the diversity of the samples when visualizing the same neuron with different initializations. To increase diversity, they add a term to randomly explore the code space. However, by just doing so, this exploration can lead to codes that gives *unnatural* images. Thus they propose to solve this issue by also learning a prior on the code space, so that the code space, even when explored , will be constrained to stay in regions of natural images. The way they train a prior on the code space is by training the

Figure 2.7: Examples of generating hybrids from Nguyen et al. (2016a). Left are images generated from a set of classes. Right are the same images hybridized with the class *candle*.

generator in an autoencoder-like fashion. The code is trained to output the image, minimize the same three losses used earlier, but also to reconstruct the code from the image, as if it was an autoencoder on the code space. One important observation was that the path from the predicted image to the reconstructed code was frozen (not trained), so that the training would constrain the input code to produce an image which necessarily have the correct code. Once the generator is trained, they can visualize a neuron of a deep neural net classifier. To visualize a neuron, they use an optimization procedure with 3 terms, a first term for going towards code spaces leading to natural images, a second term to maximize the activation of the desired neuron, and a third term for randomly exploring the code space (for diversity). The combination of the three terms thus give a diverse set of codes that lead to natural images and at the same time activate maximally the desired neuron. See Figure 2.8. As a summary, see also Figure 2.9, which shows a comparison between different activation maximization techniques.

Generative adversarial networks

Here we present some recent works using generative adversarial networks (GAN) to build generative models of images. Goodfellow et al. (2014) was the first paper about GANs, they introduced the theory and the training procedure, and listed GANs advantages and disadvantages compared to other kinds of generative models. For instance, one of the advantages of GAN is that sampling is straightforward (just a forward pass on the generator), whereas in other generative models, like Boltzmann machines (Salakhutdinov & Hinton, 2009), sampling requires a

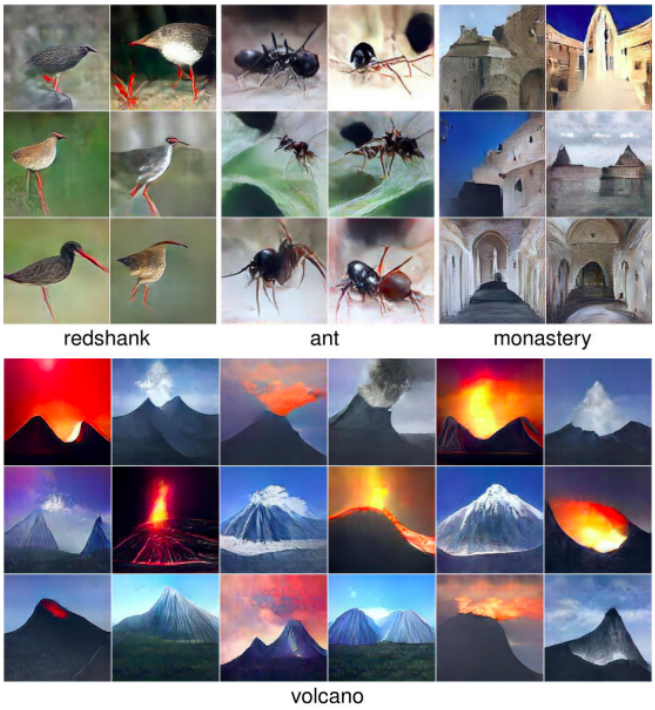


Figure 2.8: Images generated from Imagenet classes from (Nguyen et al., 2016b).

Markov chain, which requires multiple steps and thus can be much slower. Also, while GANs support arbitrary kinds of architectures for the generator and the discriminator (with the condition of being differentiable), their training is often harder and unstable, and they can fail in different ways. One way they can fail is through the *Helvetica* scenario, where the generator outputs the same image for different inputs.

The idea of GANs is to train two models simultaneously: a discriminator and a generator. The generator takes a vector of random noise (typically generated with a simple distribution like uniform or Gaussian) as input and outputs the generated sample (e.g., an image). The discriminator takes as input either a generated sample or a real one (sampled from training data) and predicts whether it is generated or real. The two models (generator and discriminator) play a game. The generator tries to generate samples which fool the discriminator, that is, it tries to make the discriminator predict that the generated data are "real". On the other hand, the discriminator tries to discriminate between real and generated data as accurately as possible. Training is done by alternating between optimization of the discriminator

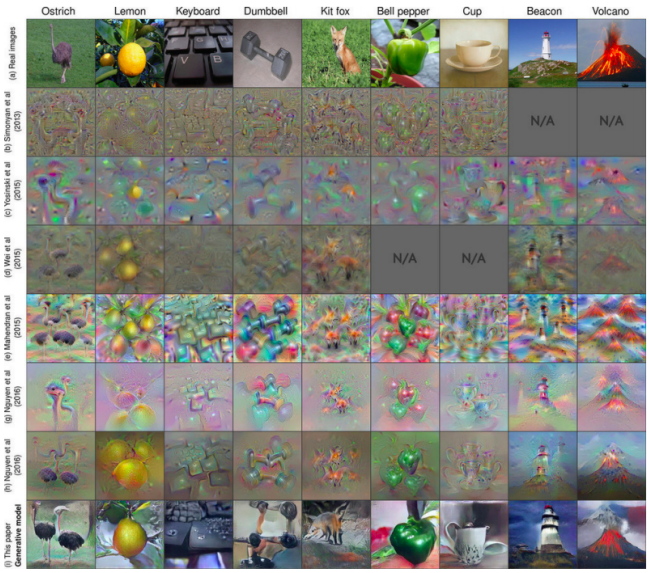


Figure 2.9: Comparison between different activation maximization techniques, from Nguyen et al. (2016a).

and optimization of the generator. In GANs, following the notation of (Odena et al., 2016), we optimize the objective

$$L = \mathbb{E}_{x \sim p_{\text{real}}}[\log P(S = \text{real}|x)] - \mathbb{E}_{x \sim p_{\text{gen}}}[\log P(S = \text{gen}|x)], \tag{2.20}$$

where p_{real} is the distribution of the real data, p_{gen} the distribution of the generated data, S is the discriminator label: either real or generated. The discriminator tries to maximize L whereas the generator tries to minimize L.

Typically, both the generator G and the discriminator D are neural networks. To sample from p_{gen} in equation 2.20, we first sample a noise vector z from a uniform distribution, typically $z_i \sim \text{Unif}(-1, 1)$ where i is the i-th dimension of the vector z. After sampling z, we pass it through the generator G to obtain an example: $x = G(z)$. The discriminator $D(x)$ predicts the probability (a scalar) of being 'real' where $x \sim p_{\text{real}}$ or $x \sim p_{\text{gen}}$.

Mirza & Osindero (2014) introduced conditional generative adversarial networks, a simple modification of the original GAN to support conditioning. In conditional GANs, the generator gets a noise vector but also a conditioning vector as inputs, for instance a one-hot vector which encodes a class that we want to condition on, and outputs an image. The discriminator gets an image but also the conditioning vector as inputs and returns a scalar between 0 and 1. Training conditional GAN is the same as the original GANs.

Denton et al. (2015) exploited the multi-scale structure of the natural images to use GANs to learn to gradually up-sample low-resolution images and transform them to a higher resolution. In their setup, they have multiple GANs, one for each scale, e.g., 8×8, 16×16, 32×32, 64×64. Starting from an image of size, e.g., 64×64, they build successive down-sampled versions of that image using a down-sampling operation. Generation starts from the lowest scale, e.g., 8×8. The GAN of each scale is trained to upscale the low resolution image. For model selection they used Parzen-likelihood estimation. For evaluation of their model, they did an experiment with humans selecting if the image is a real or a fake, comparing standard GAN and their method.

The first paper that attempted to improve the stability and performance of GANs is Radford et al. (2015). Radford et al. (2015) introduced an architecture along with optimization hyperparameters and some tricks for training efficiently a deep convolutional neural network generator. They show that their GAN can generate realistic images of size 64×64 and 32×32 on three datasets, a dataset of faces, a dataset of bedrooms (Xiao et al., 2010), and Imagenet (Deng et al., 2009). See Figure 2.10. They evaluated their model by visual inspection and by using the learned representation of the discriminator for supervised tasks. They also visualize what these models learned and show that some convolutional filters learned how to draw specific objects. They also show that the model is not memorizing the images by noticing that interpolation between images in the latent space (input of the generator) is smooth. Interestingly, they also show that it is possible to transform images by doing analogical reasoning using the latent representation of the images. They first average the latent representation that lead to smiling woman images, which can be seen as "smiling woman" representation. Then, similarly, they compute a "neutral woman representation" and "neutral man" representation. Finally, they subtract "neutral woman representation" from "smiling woman" and add "neutral man" representation, and generate an image from that representation to obtain a smiling man image. See Figure 2.11 for an example.

Dosovitskiy & Brox (2016) proposed a proxy to a human *perceptual* metric for images which use GANs. Traditionally, in autoencoder-based models the loss used between reconstructed and real images is Euclidean distance in the pixel space. However, Euclidean distance in the pixel space does not correlate with a human perceptual distance (Wang et al., 2004). They propose to use a distance on an abstract feature space, which is learned by a classifier, so that it contains semantic information about the image. However, using a loss based on a feature space alone does not result in good generated images, because the mapping from images to features is not a one to one mapping, thus many non-natural images can get mapped to the same feature vector values. So they propose to use GAN, along with the features-based loss, to constrain the generated images to be in the manifold of

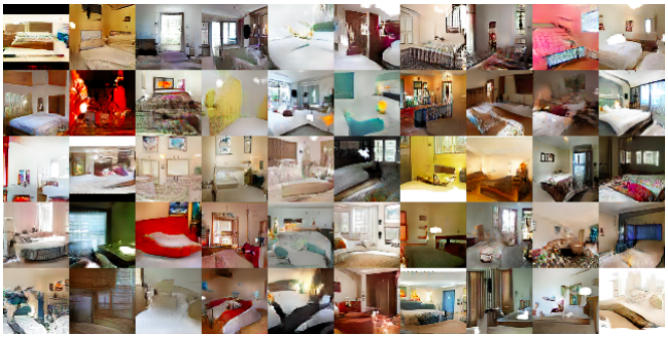


Figure 2.10: Bedrooms generated from (Radford et al., 2015).

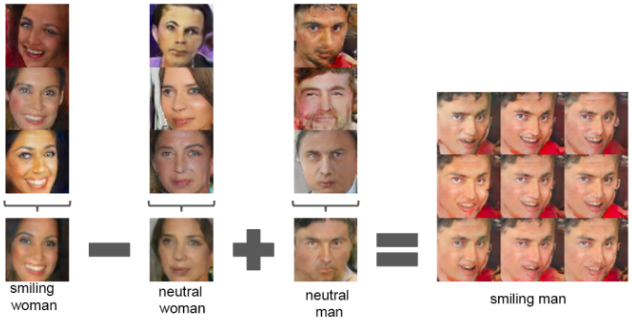


Figure 2.11: Example from (Radford et al., 2015) of using vector arithmetics on the latent representation to convert a smiling woman to a smiling man.

the natural images. They use this setup in autoencoders, which traditionally use the Euclidean distance on the pixel space, and replace it by a loss based on GAN combined with Euclidean distance in the feature space. They obtain images that are more realistic than ones obtained with the Euclidean distance on the pixel space, which are often blurry. See Figure 2.12 for an illustration of the effect of the proposed losses.

Salimans et al. (2016) proposed a set of heuristics to improve the stability of GANs, and propose for the first time to use GANs in semi-supervised learning and obtain state of the art results. One of the tricks that helped semi-supervised learning a lot is feature matching. With feature matching, instead of having a discriminator that outputs a scalar, they have a discriminator that outputs a vector of features, and the goal is turned into matching the statistics of the set of features of the real data and the generated data, the statistic which is matched is the mean of the features over the mini-batch of examples. Also, importantly, they introduced

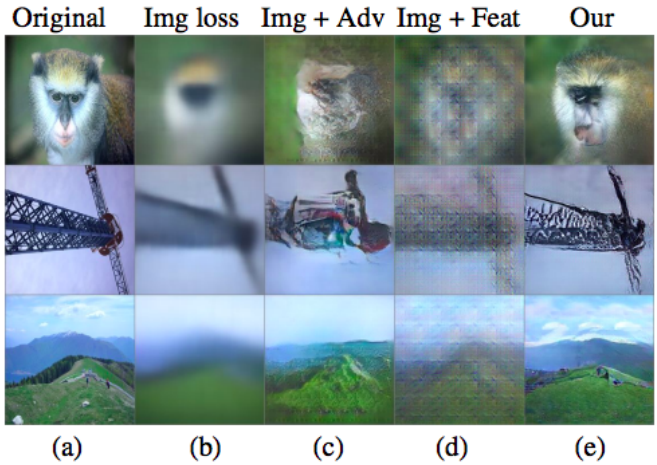


Figure 2.12: Effect of the losses used for training an autoencoder on the reconstructions. (a) refers to the original images. (b) uses the mean squared error on the pixel space. (c) combines mean squared error on the pixel space and a GAN. (d) combines mean squared error on the pixel space and feature space loss. (e) combines mean squared error on pixel space, GAN, and the feature space loss.

the *inception score*, which became a standard tool to evaluate image generative models. They found that the *inception score* correlates well with human scores of whether the images are realistic. The inception score relies on a classifier trained on Imagenet, and it is meant to measure the *objectness* of set of images, from the eye of the classifier. The objectness is high when the classifier is confident that the images belong to one of the classes (low entropy), and also high when the classes of those images are diverse (high entropy), so that not all the images belong to some dominant category of objects.

Chen et al. (2016) learn disentangled factors of variation of data in an unsupervised way with GANs. The idea of learning disentangled factors of variation dates back at least to Bengio et al. (2012). The assumption is that variability of the data comes from combining independently multiple *factors*, such as lighting, view point, and characteristics of the face. The role of unsupervised learning would then be to *disentangle* the factors from the image. In standard GAN, the latent variables (the input of the generator) are not constrained, that is, there is nothing that force them to be meaningful (or disentangled). Their proposed framework supports disentangling continuous and discrete latent variables. To avoid learning uninterpretable latent factors, they propose to decompose the input of the generator into incompressible noise and what they call latent codes. Then, they impose high

mutual information between latent codes and the generated images, which can be interpreted as a constraint to not lose information about the latent codes in the generation process. A variational formulation of the mutual information is provided and the GAN is trained as always with that additional mutual information loss. The variational formulation requires a third network (in addition to the generator and discriminator) to predict the latent codes from the images, which correspond to the interpretation that the information about the latent codes should not be lost after the generation process. They demonstrate that the GAN can learn concepts like rotation, azimuth, lightning, presence/absence of glasses, hair style, emotion, width in a completely unsupervised way. They also show that without using their mutual information objective the latent factors are non-meaningful (that is, they are *entangled*).

Reed et al. (2016) propose a framework to generate images in a controllable way, where the user can provide the location and content of different parts of the image to be generated. For instance, the user can provide as input a bounding box where a bird should be drawn, or even the location of its constituents (beak, belly, and right leg, for instance). The rest of the details can be specified using an informal text like "This bird has a black head, a long orange beak and yellow body". They use datasets where the keypoints of the constituents and text captions are provided and use them to condition the images. They also show that it is possible to learn to generate directly the keypoints of the parts of the image from informal text, so that only the informal text is needed at test time. Overall, they show that by using these additional conditioning information, it makes it easier to generate realistic and higher resolution images (128×128) than previous works.

In Zhang et al. (2016), similarly to Reed et al. (2016), they propose a way to condition images on text, but generate a much higher resolution images of size 256×256. To do that, they decompose the generation into two steps, a first step where the GAN generates a low resolution (down-sampled, e.g., 64×64) version of the image (a "sketch" containing basic shape and colors) conditioned on the text and a second step where the GAN adds the missing details in a higher resolution image (e.g., 256×256), also conditioned on the text. See Figure 2.13 for an illustration.

Zhu et al. (2016) used GANs to perform image manipulation. According to the paper, image manipulation is a well established field in computer graphics literature, but they say that when these methods fail, they tend to produce unrealistic output images. The reason is that they rely on low-level principles (pixel level). GAN can capture higher level information about images such as objects shapes. Their goal it to provide some high level sketching tools to the user, and use the GAN to keep the images "realistic". They provide a different set of image manipulation tools. The idea is that the user use the tools to provide some constraints

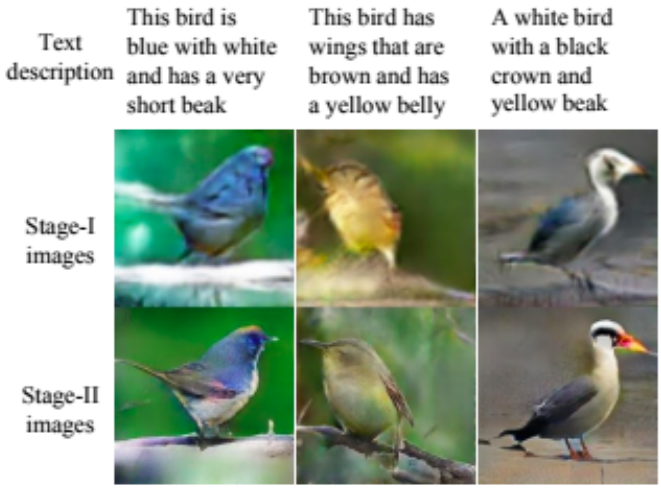


Figure 2.13: The two steps generation conditioned on informal text from Zhang et al. (2016). In the first stage, a low-resolution of the image is generated conditioned on the text. Then, in the second stage, a high-resolution of the image is generated conditioned on the low-resolution image and on the text.

on the image in some defined region (for instance, a bounding box) and the GAN is asked to satisfy the user constraints, as well as to stay in the manifold of natural images to avoid generating unrealistic images. They provide different kinds of tools like a coloring brush, sketching brush, and warping brush. For instance, the user can provide a sketch of a mountain, and the GAN can automatically add the details (e.g., textures).

Isola et al. (2016) showed the effectiveness of the GAN framework for image-to-image translation, that is, the goal is to learn a mapping from image to image, where the source image comes from a source domain and the target image comes from a target domain. For instance, it can be used to transform a picture of a house into a plan, or from a sketch into a detailed image, or from a painting style to another. The framework supposes a dataset where we have a ground truth image-to-image mapping, from the source domain to the target domain. Their work is an application of conditional GANs, where they introduce novelty in the architectural choices to make the images more realistic.

Maaløe et al. (2016) extend the framework of conditional GANs, and show that providing label information to the generator can help to generate more realistic images, especially in datasets with high variability. The way the extend the conditional GAN by forcing the discriminator to not only predict whether an image

is real or fake, but also to predict its class. The generator gets as input the noise vector and label information while the discriminator gets as input the image only. They also provide a new metric for evaluating the generated images called *discriminability*. Discriminability is the fraction of generated samples, conditioned on a given class, that are indeed recognized belonging to the conditioned class using a classifier.

Ledig et al. (2016) used GAN on the problem of super-resolution. In super-resolution, the goal is to transform an image from low resolution to high resolution (for instance, with 2x or 4x ratios). To train a model, images of high resolution are typically blurred with a Gaussian blur and resized, and the network is asked to transform the blurred images into the original high resolution image, as closely as possible. The models are usually trained with mean squared error as a loss function. In this paper, similarly to (Dosovitskiy & Brox, 2016), they propose to use GAN to define a *perceptual loss*, where they combine a GAN adversarial loss and Euclidean distance in the feature space for a chosen feature space. They demonstrate enhanced results when using their proposed perceptual loss. Figure 2.14 shows a comparison of different methods for super-resolution.

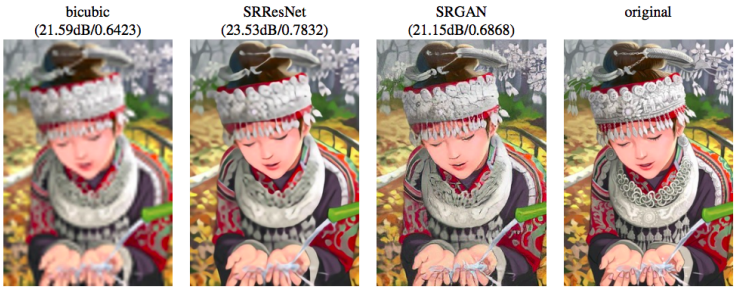


Figure 2.14: Results of 4x super-resolution obtained using different methods. The method proposed by Ledig et al. (2016) is SRResnet.

Zhu et al. (2017) generalized the image-to-image translation framework (Isola et al., 2016) to situations where pairs of image from a source domain to a target domain are not available. In this task, we are given samples from the source domain and samples from the target domain but without pairings between the two, and the goal is to learn to transform images from the source domain to the target domain. Like (Isola et al., 2016), unpaired image-to-image translation use a GAN to transform images from the source domain to the target domain. However, a standard GAN can only ensure that the distribution of the generated images of the target domain match the distribution of the real images from the target domain. That is, given an image from the source domain, there is no pressure to make the gener-

ated image (from the target domain) related to the original image (from the source domain). To ensure they are related, they propose to use *cycle consistency* losses, which constrain the generated image from the target domain, when transformed back to the source domain, to reconstruct the original image in the source domain. Also, symmetrically, they constrain the generated image from the source domain, when transformed back to the target domain, to reconstruct the original image in the target domain. See Figure 2.15 for some examples.

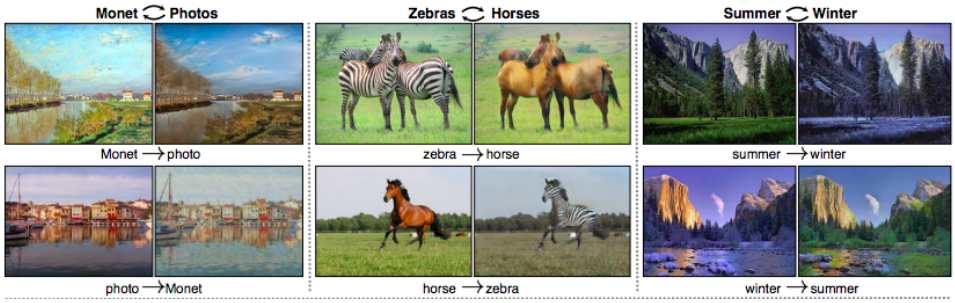


Figure 2.15: Transforming unpaired images from a source domain to a target domain using CycleGANs (Zhu et al., 2017).

Neural style transfer

Neural style transfer (Gatys et al., 2015b) is a recent framework that can be used to redraw a picture using the *style* of another picture (e.g., see Figure 2.16). It had a lot of success in the Internet sphere since the first days the paper came out, and many interests in the academic research but also in industry. It demonstrated that by just using a pre-trained convolutional neural network and a proposed *style* representation, it is possible to generate images with high quality reproducing an image with the style of another one. More concretely, we are given two images, C, the content image and S, the style image. The goal is to redraw C using the style of S, e.g., redraw the picture of a landscape with the style of Van Gogh. First, we initialize randomly a target image T, then similarly to activation maximization (see Section 2.2.5), we optimize the pixel space of T with gradient descent in such a way that it matches the *content* of C and the *style* of S. The objective function optimized on the target image T have two terms, a content loss and a style loss.

The content loss is the squared Euclidean distance between the content representation of T and C, where the content representation is chosen as one of the intermediate layers of a pre-trained convolutional neural net classifier trained on Imagenet. The goal of the content loss is to push T towards C in the feature space

defined by the convolutional neural net layer. Intuitively, the content loss reproduces high-level details of C in T, leaving the low-level details like brush strokes, to the style loss. The main contribution of the original paper is to propose the style loss. More concretely, in a convolutional layer we have several feature maps, each feature map correspond to the result of convolution of the previous layer feature maps with a filter. For a given layer, they represent the style by using the *Gram* matrix G, which is obtained by computing the correlation between all pairs of feature maps and the correlation is taken over the spatial extend. That is, the index which varies in the correlation sum is the position of the filter response. The Gram matrix G is a matrix with d^ℓ rows and d^ℓ columns where d^ℓ is the number of feature maps in layer l. For each pair of feature maps i and j, $G^\ell_{ij} = \sum_k F^\ell_{i,k} F^\ell_{j,k}$ where G^ℓ is the resulting style representation of layer l and $F^\ell_{i,k}$ is the i-th vectorized feature map of the layer l, that is, $1 \leqslant k \leqslant w^\ell h^\ell$ where w^ℓ and h^ℓ are respectively the width and height of the feature maps of the layer l. For a given layer, the style representation of T is matched to the style representation of S using the squared Euclidean distance. Notably, the style representation is optimized for several convolutional layers at the same time, not only one, in order to capture the style in several scales.

In the first publication of neural style (Gatys et al., 2015b), the Gram matrix was assumed to represent the "style", but there was no formal definition of what a style is and there was no clear explanations why the Gram matrix would represent the style. The first paper to provide a theoretical explanation was Li et al. (2017). They showed that matching the style representation is equivalent to applying Maximum-Mean-Discrepancy(MMD) (Gretton et al., 2012), which is a metric that is used originally to compare two probability distributions, and is used typically in domain transfer, to match the statistics of a source domain with a target domain. More concretely, they showed that matching the Gram matrix of S and T is equivalent to considering the feature map columns $F^\ell_{.,k}, 1 \leqslant k \leqslant w^\ell h^\ell$ as samples from a distribution, that is, for a given layer ℓ, each $F_{.,k}$ is considered as an observation, and minimizing the Euclidean squared distance between the Gram matrices is equivalent to match the distributions of the feature map columns of S and T, where the distributions of S and T are each formed by $\{F^\ell_{.,k}, 1 \leqslant k \leqslant w^\ell h^\ell\}$. Thus, as pointed out by Li et al. (2017), matching the feature map *columns* distribution is desired for style transfer because the positions of the features (in the feature map) are ignored. Intuitively, it can be seen as a way reproduce the textures of S in T independently of the position in the image.

Subsequent works have refined the original paper (Gatys et al., 2015b). Mainly, two important aspects were considered, speed and having more control on the resulting image. The original style transfer is expensive because it requires a full

optimization loop for each pair of content and style images. Thus, several papers focused on ways to make it faster. For instance, Dumoulin & Visin (2016) use a neural network which learns to predict the resulting image from the content image and a style identifier. The neural network learns from several content images and styles using the same loss function that is used in the original paper (Gatys et al., 2015b). The advantage is that once the model is trained, it can be used to predict instantly the resulting image for a new content image and one of the styles that the model was trained with, without using an expensive optimization loop. Also, some papers proposed ways to have more control on the resulting image. For instance, Champandard (2016) proposes to use *semantic maps*, which are additional annotations in the form of segmentation maps that divide an image into multiple segments, each segment highlighting a meaningful part of the image (e.g., trees, sea, mountains). The user gives an input image, its segmentation map (or use a model to predict automatically the segmentation map), then a second *desired* segmentation map (which could also either given by the user or obtained automatically using model), which will be used as a basis to draw the output image. The goal is to match regions of the desired segmentation map with regions from the input image segmentation map then use the "style" in the input image of a given region to draw the content of the output image to its matched region from the input image. Figure 2.17 from Champandard (2016) provides an example.

Auto-regressive models

Auto-regressive models are models that factor the joint probability distribution of D-dimensional inputs $p(x) = p(x_1, x_2, ..., x_D)$ into a product of conditionals in some predefined order: $p(x) = p(x_1)p(x_2|x_1)\dots p(x_D|x_1, x_2, \dots x_{D-1})$ and parametrizes each conditional term of the product. The main advantage of auto-regressive models is that contrary to other models which optimize the likelihood in an approximate way (Salakhutdinov & Hinton, 2009; Kingma & Welling, 2013), they optimize it directly and once trained, it is possible to evaluate the probability of the data without resorting to any approximation. Regarding generation, their main drawback is that they are relatively slow due to their sequential nature: each dimension have to be generated one after the other, following the predefined ordering of dimensions. Training is done using maximum likelihood,

$$\theta^* = \arg\max_\theta \sum_i \sum_j \log p_\theta(x_j = X_{i,j}|x_1 = X_{i,1}, x_2 = X_{i,2}, \dots, x_{j-1} = X_{i,j-1}) \tag{2.21}$$

where θ are the model parameters, X is the training data, and $X_{i,j}$ is the j-th feature of the i-th example. Once trained, generation is done by generating one dimension

Figure 2.16: Example of style transfer. Left image is the content image. Right image is the style image. Bottom image is the result.

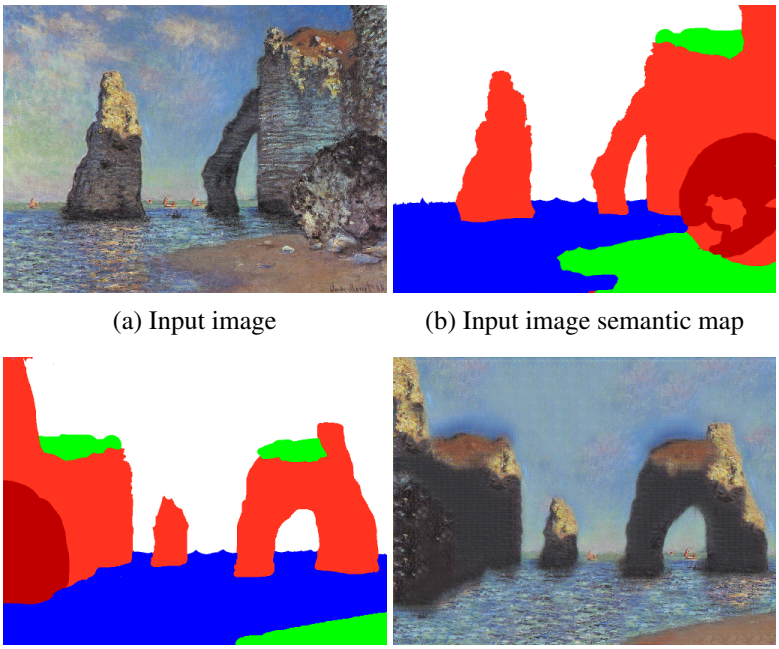
(a) Content

(b) Style

(c) Result

at a time given the previously generated ones. That is, we first generate x_1 from $p_\theta(x_1)$. Then, once generated, we use $p_\theta(x_2|x_1)$ where x_1 is replaced by the value generated previously in order to specify the conditional distribution of x_2 given x_1

Figure 2.17: Example of using semantic maps from Champandard (2016). Pictures are taken from here.



(a) Input image (b) Input image semantic map

(c) Desired semantic map (d) Output image

and generate x_2. The process is repeated for all the dimensions, adding each time a new dimension to the conditionals.

Different auto-regressive models differ by how the conditionals are parametrized and how the ordering is chosen. Examples of auto-regressive models are NADE (Larochelle & Murray, 2011), DARN Gregor et al. (2013), spatial LSTMs (Theis & Bethge, 2015). In image generative modeling, the dimensions are pixel intensities and thus each pixel is generated one at a time, following a pre-defined order. Several models have been proposed to use auto-regressive models for image generation. Theis & Bethge (2015) proposed to use recurrent neural networks (more specifically LSTMs (Hochreiter & Schmidhuber, 1997)) to model the conditionals. Notably, (Oord et al., 2016) improved state of the art of generative image modeling of auto-regressive models by proposing the PixelRNN and PixelCNN models. Their work is similar to Theis & Bethge (2015), the main difference is that the conditional probability distributions of pixel intensities are discrete, instead of continuous. This simple discretization of the pixel intensities made the training easier (than using continuous distributions), it also has the advantage of being arbitrar-

ily multi-modal without a prior on the shape of the density contrary to continuous distributions. Also, they provided some architecture innovations by using *masked* convolution, which influenced subsequent works. Masked convolution is a way to use convolutional neural networks for doing auto-regressive modeling by forbidding the model to see the future, that is, when modeling the conditional x_i given $x_1, \dots x_{i-1}$, the convolution is applied to the full image but masked in such a way that "future" pixels $x_{i+1} \dots x_D$ are not used, so that the model keep being a valid auto-regressive model. The reason masked convolution was used is driven by speed in order benefit from the ease of parallelization of the convolution operation.

van den Oord et al. (2016) extended PixelRNNs (Oord et al., 2016) with a conditional version to be able to generate from classes or learned embeddings or even to replace the decoder part of an autoencoder (see Figure 2.18). They also improved the PixelCNN architecture, making it closer in performance to PixelRNN with the advantage of being much faster to train. Similarly, (Gulrajani et al., 2016) proposes to combine variational autoencoders and PixelCNNs in such a way that latent representation of the autoencoder captures the high-level content of the image while the PixelCNN decoder captures the low-level details. The main issue with standard variational autoencoders is that the approximate posterior (decoder distribution) $q(x|z)$ used is usually a simple distribution like factorized Gaussians, which cannot represent a complex distribution like natural images. Rather than factorized Gaussians, they proposed to use PixelCNNs as the approximate posterior (decoder distribution) which is very good at modeling local correlation of nearby pixels (thus, low-level details) due to their autoregressive nature. When coupled with the latent variables $p(z|x)$ (encoder), they were observed to be complementary to the PixelCNN: the low-level content was handled by PixelCNN and the high-level content was handled by PixelCNN. They also observe that by adding more layers to PixelCNN, there is less and less incentive to use the latent variables as the PixelCNN becomes more complex and able to model $q(x|z)$ without needing z. However, adding more layers makes the generator expensive and the latent variables become less and less used, which is not desirable when the goal is to learn useful latent variables. They show that they could obtain state of the art results without needing much PixelCNN layers contrary to previous works, which shows that latent variables can be helpful and give complementary information.

Dahl et al. (2017) proposed to use PixelCNNs for image super resolution. They describe the image super-resolution problem and highlight its highly multi-modal nature, especially when converting from a very small scale where details are non-existent (e.g., 8×8). They show that the current mean squared error super-resolution based loss used in the literature is insufficient because it assumes a factorized Gaussian output with a fixed variance, which cannot model multiple outputs for the same input and thus results in a blurry output which can be seen

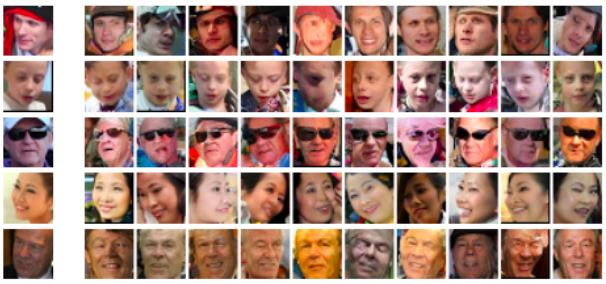


Figure 2.18: Generating new portraits of people with PixelCNN decoders (Oord et al., 2016).

as the average of all the possible outputs. Thus, they propose to use PixelCNN which not only models correlations between pixels but can also be multi-modal. They express the problem as a conditional image generation, where the output image is conditional on the low scale input image (e.g., 8×8). When using standard conditional PixelCNNs alone, they observed that the output tended to not use the conditioning information. To fix this issue, they proposed to force the model to use conditioning information by using two models, a first model that use PixelCNN to output low-level details, and a standard convolutional neural network that predicts the high-level structure and merge explicitly the information given from the two models to predict the next pixel intensity given the previous ones. See Figure 2.19 for an example.

2.3 Conclusion

In this chapter, we have reviewed machine learning and its two main branches, namely supervised and unsupervised learning. We have reviewed as well several lines of works that appeared recently in the deep learning literature where the research goal in general is how to build good generative models for generating objects that look like the training data. The generative modeling literature can be a good starting point to study systems that can generate new objects, as they explicitly rely on knowledge through the training set. However, the objective used in this literature is not well aligned with the generation of new objects since the ultimate goal is to be able to generate data that follows the distribution of the training data. Also, most papers focus on generating (known) realistic looking objects or learning representations from unlabeled data to apply them for prediction rather than discussing or formalizing new tasks related to generation. In Chapter 3, we

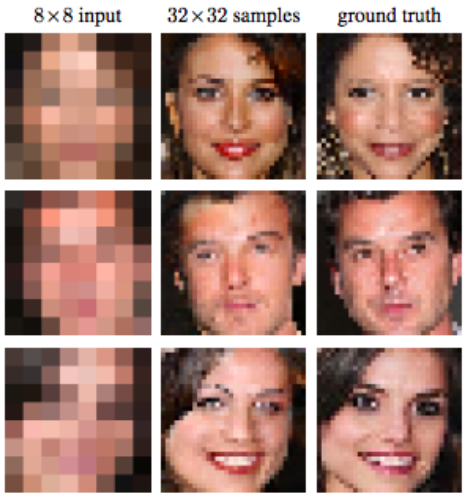


Figure 2.19: Super-resolution from 8×8 images to 32×32 images using pixel recursive super resolution (Dahl et al., 2017)

will review two fields that studied novelty generation for decades and show their current limitations. In Chapter 4, we will take a step back, and discuss about what a good knowledge-based representation can bring us for designing new objects. More concretely, we will propose a conceptual framework which explains that the current objective used in machine learning is just one among other possible objectives. We will also propose, under the proposed framework, new tasks for generative modeling, and compare them.

Chapter 3

Novelty generation

The literature of deep generative models has only recently started to work on generation. However, novelty generation has been studied for at least 20 years in the literature of design theory and computational creativity. In this chapter, we will review these fields and question their limitations. First, we will review computational creativity in Section 3.1. Then, we will review design theory in Section 3.2.

3.1 Computational creativity and genetic algorithms

Computational creativity (CC) is an interdisciplinary field whose goal is to study means of generating novelty computationally. The main questions that the field addresses are procedures/algorithms of generation, as well as evaluation metrics. Although CC is very diverse, the main fields that are studied are from art (mainly images and music; Loughran & O'Neill, 2017), although recently some authors (Loughran & O'Neill, 2017) stated that creativity should not be limited to art, and CC should target scientific fields as well. The CC field addresses both *generation*, that is, algorithms and tools that can be used to generate new objects, as well as evaluation, scores, and metrics that are used to *assess* the value and the novelty of each generated object. Most CC systems separate the generation and evaluation phases, which makes them a kind of *generate-and-test* (Togelius et al., 2011) systems. In this section we will review the field of computational creativity: Section 3.1.1, is about mechanisms used to generate objects while Section 3.1.2 is about evaluation.

3.1.1 Generation

Most CC systems are based on genetic algorithms, which we describe in Section 3.1.1. Then, we describe generative grammars in Section 3.1.1, L-systems

in Section 3.1.1, and cellular automata in Section 3.1.1. Next, in Section 3.1.1 and Section 3.1.1, we describe Markov chains and hidden Markov models. Finally, in Section 3.1.1, we describe conceptual blending, which is not exactly a tool or an algorithm of generation, but more like a general principle that have been used to generate new objects and which is common in the CC literature.

Genetic algorithms

Genetic algorithms comprise a class of optimization algorithms which mimic natural evolution. The idea of genetic algorithms came from John Holland and his team (Holland, 1992) in the 60s. The goal is to optimize a *fitness* function by evolving a population of solutions, also called *individuals*. Each individual is represented by a set of *genes*. The representation of individuals is domain dependent, for instance it can be a vector of binary values, a tree, or even a graph. The *fitness* function takes an individual as input and returns a real value indicating the desirability of that individual.

A genetic algorithm starts with a population of randomly generated individuals. First, the population is evaluated using the fitness function. Then, a subset from that population is selected to produce new individuals, called *offsprings*. Different ways of selecting individuals have been used in the literature. One common way is to randomly sample with replacement from the population. The selection probability is a monotonic function of the fitness value: the higher the fitness is, the higher the probability that an individual will be selected. Then, the sample is used to produce offsprings using two operators: mutation and crossover. Just like the representation of individuals, the mutation and crossover operators are domain dependent. Mutation takes an individual as input, perturb randomly its genes and returns a mutated version of the individual. Crossover takes two individuals as input and returns two new individuals.

In genetic algorithms, we usually distinguish two representations of the individuals, the representation where the optimization is done, called the *genotype*, and the representation where the fitness is evaluated, called the *phenotype*. When the genotype and phenotype are not the same, we need a function that maps from the genotype to the phenotype. The simplest way to represent genotypes is by using binary strings. If the genotype is a binary string, mutation can be done by randomly flipping the bits of the string with some chosen probability (e.g., 0.5). For instance if an individual represented as the binary string 0010 it can be mutated into 1011 if the first and last characters are mutated (flipped). For crossover, a position in the string is randomly chosen to split both the individuals into two chunks then the chunks are combined (concatenated) to build two new individuals, called *offsprings*. For instance, if the two input individuals are 0010 and 1101 and we

choose to split them in the position 2 of the binary string, the left and right chunk of the first individual are respectively 00 and 10. Similarly, the left and right chunk of the second individual are respectively 11 and 01. The first offspring is obtained by concatenating the left chunk of the first individual and the right chunk of the second individual: 00 and 01 which gives the first offspring 0001. Similarly, the second offspring is obtained by concatenating the left chunk of the second individual with the right chunk of the first individual: 11 and 10 which gives 1110. Once a fixed number of offsprings are created, they replace the population and the same process is repeated again for as many iterations as desired. The algorithm is stopped typically after some number of iterations or when the best fitness value does not improve anymore.

Most CC systems are based on genetic algorithms with problem specific genotypes and phenotypes. Thus, the generation of new objects in that case is done through the mutation and crossover operations which are problem specific and the population of objects is evolved using the fitness (evaluation) function.

NEAT (Stanley & Miikkulainen, 2002) proposed to evolve neural networks, which can be also be seen as a computation graph that maps an input vector to an output vector. NEAT has mostly been applied to optimize neural networks for control tasks (Wang et al., 2003). NEAT evolve the weights as well as the structure of the neural network, starting by an empty or small neural network and incrementally adding new nodes or new connections. One issue with NEAT is that the optimization becomes hard as the neural network become larger due to the large number of weights and nodes. HyperNEAT (Gauci & Stanley, 2010) proposed to extend NEAT to allow encoding the connectivity between neurons through another higher level neural network, which itself *predicts* the weights of the lower-level neural network, making it easier to express large neural network connectivity patterns with fewer parameters.

Several image generation papers in the literature use the Compositional Pattern Producing Networks (CPPN) (Stanley, 2007), which are neural networks that are evolved with genetic algorithms and which use activation functions that lead to outputs that have patterns that can be found in nature like repetition and symmetry. Concretely, when applied to images, CPPN is a neural network that takes as input two real numbers (x, y) which represent the position in the 2D image, and returns the pixel intensity (or RGB colors) corresponding to the position. By using activation functions like sine, cosine, Gaussian and sigmoid, it can render images which have the desired repetition and symmetry patterns. The images are rendered by looping through all possible (x, y) for some chosen discretization of the 2D canvas. The weights and the architecture of the neural network are evolved with NEAT (Stanley & Miikkulainen, 2002) and typically evolved in a way that it produces an image which maximize some user-defined fitness function, for instance

reconstructing an image.

Several works proposed extensions to simple genetic algorithms, in order to mimic better the biological evolution. In co-evolution, both the problem space and the solution space co-evolve, each one providing feedback to each other, rather than fixing the problem space at once. Greenfield (2008) applied it to art. McCormack & Bown (2009) proposed niche construction, which is a way to increase diversity of the solutions by splitting the population into different independent niches. They proposed a system with a swarm-like behavior. In simple evolutionary systems there is no interaction within the population. That is, they are independently evaluated for the fitness. The proposed swarm behavior adds the interaction part between agents, e.g., like in ant colonies.

(Machado & Amaro, 2013) proposed an ant colony simulator for rendering images and optimizing the parameters of the simulator using a genetic algorithm. Different fitness functions are used, among them is a combination of reconstruction error of a given input image combined with a compressibility measure (using JPEG compression) to favor simple drawings. The simulator of ants consist in a set of simulated ants that live in the 2D image canvas and that can deposit ink (which is a circle) of a certain color which they are born with and a certain size which depends on their energy (the energy can vary during the simulation). The system starts with an empty black 2D canvas and an initial set of ants at different locations of the canvas. The behavior of the ants is stochastic and depends on the parameters of the simulator. In addition to ink deposit, the ants can die or generate offsprings and their movement depends on the pixel intensities in the input image in their neighborhood.

Generative grammars

Grammars (Chomsky, 1959) are rules that define the set of correct sentences from a language. Grammars have been used initially to model natural language, but had many other applications as well, such as in programming languages and in music. There exist different kinds of grammars, among them there are the grammars from Chomsky's hierarchy, which are four classes of grammars organized in a hierarchy from the least generative power to the most generative power, where the generative power is measured by the cardinality of the number of correct sentences they can recognize/accept. In Chomsky's hierarchy, the four classes of grammars, from the least powerful to the most powerful are called: regular, context-free, context-sensitive, and recursively enumerable. The regular language is used mostly in lexical analysis to detect valid lexemes (e.g., using regular expressions). The context-free and context-sensitive languages are widely used to define the syntax of programming languages, and to recognize and parse programs. The recursively enu-

merable is the most general grammar and is as powerful as a Turing machine, but is less used in practice. All the grammars can be generative, because the rules can be exploited to generate correct sentences from the language they define.

Generally, grammars are defined in terms a of a set V of terminal symbols and a set of Σ of non-terminal symbols, and valid sentences are constructed by concatenating terminal symbols in a constrained way (not all possible combinations of terminal symbols are valid to form a sentence) where the constraints are encoded using the *production rules*. Formally, grammars are defined as a quadruplet (V, Σ, R, S) where V is a set of nonterminal symbols, Σ is a set of terminal symbols, R a set of production rules, and $S \in V$ is the starting symbol. In the general case, each rule production is written in the form of $\alpha_1, \alpha_2, \ldots \alpha_N \rightarrow \beta_1, \beta_2, \ldots \beta_M$, where the left part is called the left-hand side and the right part the right-hand side and both sides refer to a concatenation of terminal and/or nonterminal symbols, thus $a_i \in V \cap \Sigma, i = 1 \ldots N$ and $b_j \in V \cap \Sigma, j = 1 \ldots M$. The generation starts using the starting symbol S, and each time we encounter the left-hand side of a production rule, we replace it by the right-hand side and repeat the process recursively. In context-free grammars, each production rule $r \in R$ is in the form of $\alpha \rightarrow \beta_1\beta_2 \ldots \beta_{N(r)}$ where $\alpha \in V$ and $\beta_i \in (V \cup \Sigma), i = 1, \ldots, N(r)$. α is called the left-hand side of r, and $\beta_1\beta_2 \ldots \beta_{N(r)}$ is called the right-hand side of r and $N(r)$ is the number of components of the right-hand side of r. To generate a sentence from the language, we begin by the starting symbol S then we select a production rule $r \in R$ for which the left-hand side is the current symbol, S. Then we expand, that is, we replace the current symbol by the right-hand side of the selected rule. The right-hand side is a concatenation of terminal and/or nonterminal symbols ($\beta_1\beta_2 \ldots \beta_{N(r)}$), $\beta_i \in (V \cup \Sigma)$. We then recursively expand each nonterminal symbol by choosing a production rule and replace each nonterminal symbol by the right-hand side of the production rule until we are only left with terminal symbols.

During generation, if several rules have the same left-hand side (thus the same non-terminal symbols can be replaced by different right-hand sides in different rules), we have non-determinism, and we need to choose one of the possible right-hand sides. When we assign conditional probabilities to the right-hand sides given a left-hand side, we call them *probabilistic grammars*. Probabilistic grammars can be used to generate novel sentences from a user-defined grammar by either setting manually the probabilities or estimating them for a corpus. Grammars have been extensively been used in music generation (Holtzman, 1981; McCormack, 1996; Johnson-Laird, 2002), where for instance the musician can define a grammar for a musical genre and generate novel musics from that genre. Grammars have been

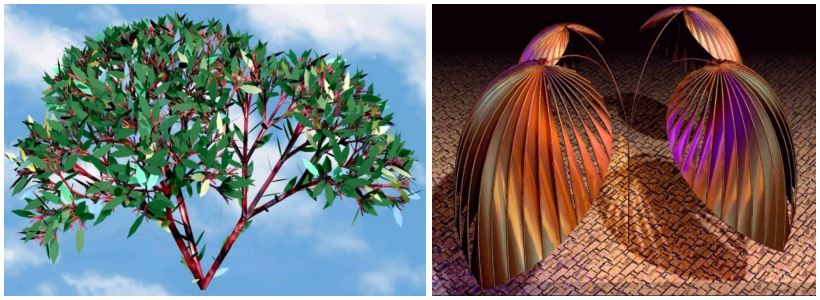


Figure 3.1: 3D models of plants generated using L-systems, using the software L-PARSER(http://laurenslapre.nl/ViewLparser/ViewLparser.html)

also used for text generation, e.g. for poetry, story telling, parodies[1][2][3]. Grammar rules themselves can be learned from a corpus, and then used to generate novel instances that resembles the ones in the corpus. The task of constructing the rules of a grammar from data is called grammar induction (Nevill-Manning et al., 1994).

L-systems

L-systems are another type of formal grammars invented by the biologist Aristid Lindenmayer and were initially used to model the growth process of plants (Prusinkiewicz & Lindenmayer, 2012). L-systems are formally similar to grammars of Chomsky's hierarchy, the difference is in how production rules are applied. In formal grammars of Chomsky's hierarchy, in each step, given the current constructed sentence, only one substring of the sentence matching the left-hand side of a rule is replaced by its corresponding right-hand side. In L-systems, all possible matches are all replaced in parallel in one step. As a result of the parallelism, L-systems can more easily be used to generate fractal structures. L-systems have been used in computer graphics to model and generate plants (Prusinkiewicz & Lindenmayer, 2012) (see Figure 3.1), and other structures in 2D and 3D (Lienhard et al., 2017), or cities (Parish & Müller, 2001). They have also been used for music generation (Worth & Stepney, 2005).

[1] The postmodernism generator: https://en.wikipedia.org/wiki/Postmodernism_Generator

[2] SCIgen: https://en.wikipedia.org/wiki/SCIgen

[3] Chomsky bot: http://rubberducky.org/cgi-bin/chomsky.pl

Cellular automata

Cellular automata (Wolfram, 1994) are discrete dynamic systems that were studied initially in the context of building self-replicating systems by Von Neumann and his colleagues. In cellular automata, the world is divided into cells in a grid (1D, 2D, or 3D grid), and the cells states are evolved using simple local rules. Each cell next state is based on the current state of the neighbouring cells and the new state is updated according to a rule. The most well known example of a cellular automaton is the game of life from John Conway (Conway, 1970), which even with its very simple rules could produce complex patterns and has been proven to be as powerful as a Turing machine (Cook, 2004), in other words it is capable of simulating a Turing machine. In the context of computational creativity, cellular automata have been used to generate visual patterns (Krawczyk, 2002) as well as music (Burraston & Edmonds, 2005).

Markov chains

Markov chains (Chung, 1967) have been used in the context of CC for generating sequences, mostly of music (Pachet, 2012). Markov chains are a stochastic process which satisfy the Markov property, which means that it is a sequence of random variables $X_1, X_2, \dots X_t \dots X_T$ indexed by a "time" component which satisfy $P(X_i|X_1 \dots X_{i-1}) = P(X_i|X_{i-1})$. Because of this property, the stochastic process can be seen as memoryless because all the previous variables except the previous one are ignored. There are also generalizations of the Markovian property that admit more than one variable in the conditionals, which instead satisfy $P(X_i|X_1 \dots X_{i-1}) = P(X_i|X_{i-1}, X_{i-2}, \dots, X_{i-d})$ where d is called the *order* of the Markov chain. When modeling a sequence of discrete variables like in music, learning a corpus can be done simply by counting the number of times each symbol is present given a sequence of d symbols preceding it, to estimate the conditional probabilities. Generation is done by greedily generating from the conditionals, one symbol at a time, and incorporating the previously d generated symbols into the conditionals. For music, it can be used to learn from a corpus of music of a certain style to generate novel variations, where the variation occurs due to the stochastic nature of the Markov chains in the generation phase. Their main advantage is the ease of learning ("learning" is just counting) and ease of generation (a simple greedy algorithm). One disadvantage is the Markovian property, which means in the context of music that the generated music cannot have long-range dependencies in time and thus no global structure by definition. Another disadvantage pointed out by Pachet & Roy (2011) is their non-controllability in the sense it is hard to have any hard constraints on the generated notes as it is often needed in music. For

that reason, several works like (Pachet & Roy, 2011) extended Markov chains to allow them to satisfy hard musical constraints.

Hidden Markov models

Hidden markov models (HMMs) are models of sequential data. HMMs assume that there exist a hidden discrete state of the system that changes in each time step H_t and that the observed data is generated by a conditional model $P(X_t|H_t)$. In HMMs, the discrete hidden states possess the Markovian assumption, that is, $P(H_t|H_1, \dots H_{t-1}) = P(H_t|H_{t-1})$. Thus, HMMs are parametrized by two conditional probabilities, the hidden state update $P(H_t|H_{t-1})$ and the conditional probabilities of the observed data $P(X_t|H_t)$ which only depends on the current hidden state. It can be shown that although in HMMs the Markovian assumption holds (by definition) in the hidden state, it does not hold in the observed data, thus $P(X_t|X_1 \dots X_{t-1}) \neq P(X_t|X_{t-1})$. HMMs are learned from a corpus of sequential data using maximum likelihood with the EM algorithm (Eddy, 1996). HMMs have been widely used in speech recognition and handwritten text recognition. In the context of computational creativity (CC), they have been used for music generation (Farbood & Schöner, 2001; Van Der Merwe & Schulze, 2011).

Conceptual blending

Several CC papers are based on the idea of conceptual blending, which goes back to 1993. It is defined by Fauconnier & Turner (1996) as: *"a basic mental operation that leads to new meaning, global insight, and conceptual compressions useful for memory and manipulation of otherwise diffuse ranges of meaning"*. Conceptual blending is believed to explain, among other things, creative thinking. It has four components, two input spaces, a generic space, and the blended space. The two input spaces are each a space of concepts we want to match, while the generic space represents what is common between the two input spaces. The mental operation of blending consists in merging the two input spaces into a blended space, providing in the blended space a new meaning, that is not available in the original two input spaces. Blending consists in partially matching concepts (that is, matching only a subset of concepts with each other) in the first input space, with concepts in the second input space. The result of that operation is that matching two not necessarily related concepts from two input spaces can give rise to a new meaning in the blended space. One typical example they give in the paper is how we can create a new meaning by combining two not necessarily related words in natural language. For instance, the word "land yacht" contains the word "land" and the word "yacht". These are not necessarily related concepts, because a yacht is supposed to float in

water not land. The input space of concepts related to the word *land* could be listed as: driver, road, car, owner. The input space of concepts related to the word *yacht* could be listed as: water, skipper, course, tycoon. In blending, matching consist in matching concepts from the two input spaces, e.g., we can associate water to land, skipper to driver, course to road, yacht to car, tycoon to owner.

3.1.2 Evaluation

Evaluation metrics for detecting new and valuable objects are recognized in CC as one of the main issues and are an open research question. Several evaluation metrics have been proposed in the literature.

Hand-designed evaluation

This review on hand-designed evaluation functions is based on (Galanter, 2012b). Back in 1933, Birkhoff proposed an *aesthetic* measure (Birkhoff, 1933). For a given object (e.g., a painting), Birkhoff aesthetic measure can be written as $M = \frac{O}{C}$ where O is the order and C is complexity. Order is the degree of unconscious tension after perceiving the object. This tension is believed to be released after perceiving features such as repetition, similarity, contrast, equality, symmetry, balance, and sequence. Complexity is amount of effort to perceive something. Zipf metrics, based on the Zipf probability distribution were used in music to detect pleasant music (Manaris et al., 2003, 2005). Fractalness measures (Taylor, 2006) were used to assess the complexity of images. Datta et al. (2006) proposed a set of 56 image features based on insights from a design course that correlated well with human ratings of images. Supervised machine learning models have been used to learn how to compute music aesthetic rate (Manaris et al., 2005) (neural networks), but also for discriminating between different styles of music in (Machado et al., 2004)(neural networks) and in Bergstra et al. (2006)(Adaboost), or different styles of paintings (Machado et al., 2008)(neural networks).

Several works use evolutionary systems (e.g., genetic algorithms) to generate art, although the main issue is the choice of the fitness function. One possibility used in literature is to put human in the loop. For human in the loop, some works use individuals or sometimes crowdsourcing. When individual people are used as evaluation the result can lead to skews like fatigue, boredom, etc. Crowdsourcing can solve the previous problems but can lead to a sort of "average" solutions, which do not reflect any *uniqueness* or individuality, as this would happen with an artist. An alternative to using human in the loop is to design fitness functions manually.

McCormack & Bown (2009) distinguishes several kinds of fitness functions. The first kind is fitness functions that are based on performance goals. For in-

stance, Sim's creatures (Sims, 1994) propose selection pressure (e.g., walk, run, jump) for evolving a creature in a simulated physical world. The second kind is fitness functions which are expressed as an error relative to a set of exemplars, for instance by checking how much the designed painting is close in pixel space to a painting, or play music by mimicking sounds of acoustic instruments. The third kind is fitness functions based on complexity measures. Machado & Cardoso (2002) implemented Birkhoff complexity and used *jpeg* compressibility as a proxy to complexity, while fractal compressibility correspond to the observer "processing complexity", which is the order. Moles (1968) used a complexity based on information theory, namely entropy, where low entropy means high compressibility. However, it has been observed that entropy used *as is* may not match our judgments because the most complex objects would be random noise. Objects that we encounter and classify as complex are not random. Thus, they proposed *effective complexity* (Gell-Mann, 2002), which is a balance between order and disorder. It peaks when the balance between order and disorder is good, and decrease after there is too much disorder (randomness).

Theory of curiosity

Schmidhuber (2010) proposes a theory of curiosity using data compression that can explain various phenomena such as artistic work (music, paintings), jokes, and scientific discovery. The main idea of the paper is that an AI agent should have an intrinsic reward for choosing actions which will lead itself to a better understanding of the world, measured by compression progress of the data it has seen in its history. The AI agent is not interested in actions that lead to observed data that it can already compress well, nor data that is incompressible (random), but data that has new (relative to the current compressor) patterns to compress. The paper discusses two concepts, beauty and interestingness. Beauty is the compressibility of a pattern (e.g., how much the agent can recognize an image or music): the more compressible a pattern is (e.g., an image with a lot of symmetries and repetition), the more it is considered beautiful. However, the AI agent can become bored if it only sees patterns that it can compress well, hence the proposed intrinsic motivation that leads to new kinds of compressible patterns, which is guided by interestingness. *Interestingness* is the *derivative* of beauty with respect to time. In other words, what is really sought by the agent is the compression *progress* of its own historical data through time. For instance, a scientific discovery like the Newton laws of motion has the ability to compress data such as the movement of planets or ordinary objects that we see in our daily lives with very simple formulas. Thus, these simple formulas led to a big compression progress in the field of mechanics.

Liapis et al. (2013) proposed an implementation of Schmidhuber's theory of creativity (Schmidhuber, 2010) by using compositional pattern producing networks (CPPN; Stanley, 2007) as generators, denoising autoencoders (Vincent et al., 2010) as compressors, and novelty search (Lehman & Stanley, 2011a) as a principle for exploration of the space of objects. The algorithm consists in alternating between two phases, the transformation phase and the exploration phase. In the transformation phase, a denoising autoencoder is trained on the currently available objects (e.g., images). The denoising autoencoder is meant to compress the available data by recognizing the repeated patterns in the data and encoding them into a high-level representation. Once the autoencoder is trained, its learned representation is used as a basis for exploration of new objects in the exploration phase. In the exploration phase, CPPN are used to generate objects and the weights and architecture of the CPPNs are evolved using novelty search as a fitness function, which seeks for objects that are different from the ones we already have using the Euclidean distance based on the learned representation of the autoencoder, so that the distance metric is applied on high-level features learned by that autoencoder. They apply their system to the spaceship images generation in games, which consist in black and white sprites of size 49×49. The process starts with some initial spaceships and they alternate between the transformation phase and the exploration for several iterations, where in the transformation phase the best (according to the novelty criterion) generated objects in the exploration phase are used to train the denoising autoencoder and the exploration phase uses the previously trained autoencoder to measure novelty of newly generated objects. See Figure 3.2 for an example of generated spaceships.

Novelty search

Lehman & Stanley (2011a,b) propose to abandon objectives/fitness functions by arguing that traditional optimization does not reward stepping stones, that is, exploration steps for which the solutions are momentarily worse but can lead to better solutions in the long run. Their main statement is that when stepping stones are ignored, it is more likely to be trapped in local minima. Instead of hoping to discover new kinds of objects by using optimization, they propose to directly use novelty as an objective, and ignore the fitness function. The search of novelty paradoxically leads to better solutions (according to the fitness) than traditional direct optimization of the fitness in two experimented cases: two-dimensional robot maze and three dimensional biped locomotion. Novelty search works through a fitness function which rewards solutions with novel behaviors. They measure novelty of a solution with respect to past solutions whose behaviors were highly novel. They characterize novelty in the behavior space, that is, not in genes, but their actual be-

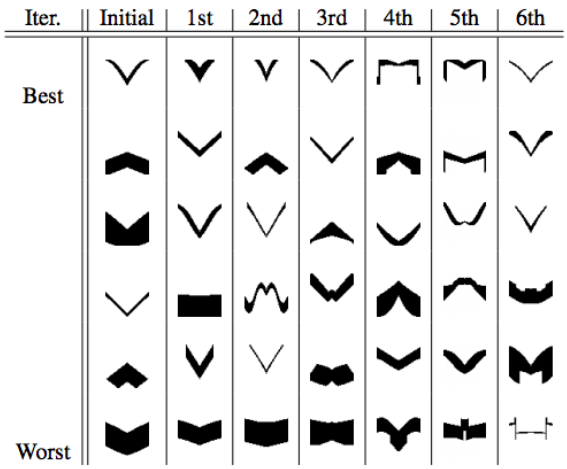


Figure 3.2: Generated spaceships from (Liapis et al., 2013). The columns refers to iterations. In each iteration we generate new objects, relative to the previously generated objects. See the text for more details.

havior when put in the real world (or simulation). Novelty of a solution is measured as the sparseness of its surrounding solutions by computing the average Euclidean distance of the vector characterizing the behavior space to its k nearest neighbors. The bigger it is, the most novel the solution is considered. When the average distance to the k nearest neighbors is beyond some threshold, the solution is added to a permanent archive, which is used to evaluate the novelty of the next solutions. They explain that novelty search is not the same as random search, because it maximizes novelty explicitly relative to old solutions, that is, it explores sparse regions of the behavior space, whereas random search is stateless.

Intentionality

Apart from evaluation, another important theme in CC is intentionality, or how to make programs have intentions. Cook & Colton (2015) pose the issue of randomness in the generative systems that exist in CC, which they consider as lacking subjectivity coming from the own experience of the generator. The issue is that even if the generated objects are novel and valuable, people would not label them as "creative" if the decisions that led to the objects are based on pure randomness. They propose a simple setup of a system choosing between preferences, e.g., between different colors. The system generates comparator functions as a code to sort a list, and use that comparator to select a preference. They define a meta fitness

function for comparators and use genetic algorithms to optimize the comparators. The main idea is that even if the system as a whole is still based on randomness and hand-designed fitness functions, the stochastic part is done at a higher level, that is, once a good comparator function is chosen (based on the genetic algorithm), the system can be considered as having a subjective way of choosing between alternatives that is not based on randomness.

Ventura (2016) proposes to characterize creative systems into three axes. Their capacity of generating novel objects, their capacity to generate valuable objects, their capacity to have intentions (decisions are not random or not based on hand-designed rules) and proposes a set of simple algorithms to explain the difference between algorithms that would be considered to have intentionality and those that would not. In particular, he explains that in a generate-and-test system, where the generation and fitness function evaluation are separated, the system can explain why a given object has value, because it has access to the fitness function. However, it cannot explain how the object was generated because the process of generation is random and separated from evaluation. Contrary to a *generate-and-test* system, he suggests that a system which embeds the fitness function directly into the algorithm of generation has better chance to explain the process that led to the generation of a valuable object.

Guckelsberger et al. (2017) address the question of how a system can be truly autonomous and select its own goals. They state that the behavior of agents is relative to their physical existence/environment/social context, rather than void. The agents' primary goal is to maintain their existence (which is a hard constraint); creativity can emerge when they encounter novel situations that may end their existence. In those novel situations they have to act in a novel way. Similarly to (Schmidhuber, 2010), they also mention intrinsic motivation as a reward for doing experiments in their environments that lead to new situations while still making sure these new situations are not too much different from the known ones.

3.1.3 Conclusion

In this section, we have reviewed evaluation and generation mechanisms used typically in the computational creativity (CC) literature. As we saw, the main open question is evaluation, or the design of fitness functions, which is usually fixed by the designer of the system. Moreover, those systems usually start from "scratch", not incorporating knowledge about the domain, or incorporating knowledge manually (hard-encoded). That is, knowledge is not driven by experience or data, and there is no *representation learning* of the objects. Additionally, most systems which they use are a kind of *generate-and-test* ones, where the generation mechanism is separated from the evaluation mechanism. This makes the genera-

tion of valuable objects very slow because the generation mechanism is completely blind with respect to the fitness function. Thus, we have established in this chapter that the majority of work in CC relies on hand coding the representation and the fitness function, which means that the generated objects will reflect the choices of the programmer of the system rather than the choices of the program itself.

3.2 Design theory

There is a large body of literature about design and theories of design whose goal is to explain how the process of designing new objects (e.g., a new type of car, a new drug or an artwork) works. Studying the process of design in general, that is, independently of the subject or domain, is important given the economic and social impact of design in contemporary societies. In this section, we will explain what design is in general, present the evolution of design theories and describe a particular design theory called C-K theory. This theory has the particularity of highlighting the importance of knowledge in the creation of new concepts which can lead to new types of objects. The core notions of the theory are illustrative of the kind of concepts we can seek in creative agents seeking to generate novelty. Moreover, although C-K theory proposes mathematical theories such as propositional logic or set theory to model and represent knowledge, in principle it does not have restrictions in what formalism to use for representing knowledge. This means that the use of deep generative models as a knowledge model can also be seen as a contribution to the design literature. We shall therefore compare it to other formalisms proposed to describe C-K and explain its implications.

The section is structured as follows. In Section 3.2.1, we explain what design is, in general. In Section 3.2.3, we explain the history of the design literature and various attempts to model the process of design. In Section 3.2.4, we define C-K theory. In Section 3.2.5 we describe the existing formalisms that are used to model the knowledge, and in Section 3.2.6 we propose the connectionist formalism, a formalism of knowledge based on a machine learning.

3.2.1 Design as a fundamental human endeavor

What is common between objects such as a car, a television, a plane, a laptop, or artworks such as paintings, music, a novel, or even scientific theories such as Einstein's general relativity? In some sense, they are all objects *designed* by a person or group of persons for satisfying certain needs, sometimes even creating a new need that did not exist before. Design is an important activity of human thinking that is involved in the creation of new objects.

Before we would attempt to study the process of design, one might wonder why would we even need design in the first place? In industry, design can constitute the spirit of a company. For example, for an automobile company, designing new cars is a necessity to attract clients and make profit. The same is true in pharmaceutics where designing new drugs is mandatory to cure diseases and also because living organisms like bacteria or viruses can adapt to the available treatments. Design is also at the heart of the creation of artworks, such as music, paintings, writings, and sculptures. From these examples, we might say that design is needed because there is a certain need (either from the designer, or from the users/clients/public, or both) to create new objects, and the already existing objects do not satisfy those needs. In other situations, the needs might be implicit (that is, not being explicitly formulated by the users/clients/public), or might not even exist yet. The role of the designers is to imagine a new kinds of objects and find ways to build them (Kazakçı, 2013).

3.2.2 Design vs optimization and search

In the literature of design theory, design is often contrasted with optimization. In optimization, although the best solution is not known a priori and is *new* once it is found, the definition of the set of possible solutions is already specified and known a priori. Thus, in optimization, we do not create a definition of new object, we search for instantiations of objects that are already well defined. One useful analogy we can find in mathematics (Hatchuel et al., 2013) is that finding a solution for a system of equations is optimization, whereas defining the set of complex numbers on the top of real numbers by adding a new dimension is design. In the former, once we define the system of equations, we already know the *form* of the solution, e.g., a vector of real numbers that will satisfy linear constraints, although we do not know the optimal values of the vector of the optimal solution in advance. In the latter, we start by defining a concept by asking the question "can a number whose square is negative exist?", which leads to the creation of a new set of numbers, built on the top of real numbers, by extending the definition of numbers to have an "imaginary" dimension in addition to their "real" dimension.

In the literature of design, there is no universal definition and theoretical framework agreed upon where it is possible to study design rigorously regardless of the specificities of each field (e.g., art, architecture, engineering, science), as it is the case of decision theory. One might ask, would a theory of design make sense? Why would we need a theory where we can study the process of design, regardless of the specificities of each field? First, having a theory of design can help us understand the process better, and thus find ways to improve it. Also, having a unified theory of design can help us extract what is common between the different

fields and study these fields in a common and agreed upon framework. C-K design theory offers such a framework that has been proposed two decades ago and that generated a long thread of research on these questions. In the following, we give more background on design theory and C-K, and explain more formally the main concepts of C-K.

3.2.3 A short review of design theory literature

Among the many works in design theory, the contribution of Herbert Simon with his book "The sciences of the artificial" (Simon, 1969) remains foundational. Simon's view was that design is ultimately a problem solving activity, which led him to consider a designer as a "General Problem Solver" (Simon, 1969). The General Problem Solver was seen as computer program with a well-defined search space and desired objective to reach. The search space can be defined in terms of a graph where nodes are states and edges are transitions between states, and the objective represents one of the states we want to reach. Simon (Simon, 1969) imagines his General Problem Solver as an agent that solves a maze. Rittel & Webber (1973) coined the term *wicked problems* to describe problems for which the formulation and evaluation are ill-defined and not definite. They contrasted wicked problems to *tame* problems that one can find in mathematics, chess, or puzzles. Buchanan (1992) stressed the lack of a theory of design and showed the similarities between design problems and wicked problems. He argues that the "wickedness" of design problems comes from the fact that "*[T]he problem for designers is to conceive and plan what does not yet exist, and this occurs in the context of the indeterminacy of wicked problems, before the final result is known*". *General Design Theory* (GDT) (Yoshikawa, 1981; Takeda et al., 1990), originally proposed in 1981, is a theory of the design process for which the goal was to provide a formal and computable way to implement the design process, for instance in a CAD (Computer-Aided Design) software. GDT considers two spaces, the function space and the structure space. The specification of the object to be designed belongs to the function space, while the solution of the design problem belongs to the structure space. The design process they describe consists in a gradual mapping from the function space to the structure space.

Beyond the purely theoretical models, attempts have been made to describe design using logical or symbolic AI concepts - especially, to become able to implement it in computers. For instance, using symbolic AI, design was explained as a co-evolution of the function space (F) and the structure space (S). The function space is used to describe the functions that are expected from the objects, while the structures define how one would implement those functions. Based on those ideas, numerous computer-aided design softwares were proposed (Takeda et al.,

1990) for specific fields. At the same time, in the engineering community, design is generally seen as an evolution of the specifications, which start by an ill-posed, incomplete definition of the solution that is refined and completed by consulting the clients/users, for instance.

While these two related views of design, one from AI and one from engineering, might be considered sufficient, they both have drawbacks. Both views highlight that design starts with a *partially defined* concept, which is refined and transformed later. However, they do not explain the *mechanism* by which the specifications evolve. Somehow the most significant aspect of a design process - the creativity - is also left out by those theories, since they do not even attempt to explain it. As we shall see in the next section, C-K design theory propose that concepts (or specifications in the two previous views) come from a knowledge space and that the interaction between knowledge and novel concepts is a fundamental aspect of design .

3.2.4 C-K theory

Concept-Knowledge (C-K) theory (Hatchuel & Weil, 2009, 2003, 2007; Hatchuel et al., 2013) model the process of reasoning in design. In C-K, the process of design starts with a *concept*, called C_0. For instance, the designer might start to work on a concept of *flying ship*[4]. The particularity of concepts in C-K is that, at the beginning of design, a concept cannot be instantiated. Said in other terms, it cannot be verified beforehand that an instance exists or that it cannot exist at all. It is exactly the role of the design process to arrive at a well-defined object whose existence is either verified or denied, that is, an object which can be *constructed* (not necessarily in the physical sense). For the "flying ship" example, this is a concept in terms of C-K, since the association of the attribute "flying" to "ship" is *unusual*, and ships are objects that are supposed to float in water and are not supposed to fly. At least in the beginning of the design process, it is likely that the designer is not aware of such a ship, nor can she state with certainty that it would never be possible. Once such a concept is generated, the designer have to refine it, that is, add new properties (or attributes) to the concept, to specify more precisely the instances it can contain. For instance, the question "can a flying ship exist" can be transformed into "can a flying ship *with wings* exist?", because we *know* that (some) *flying* beings or objects have wings.

More formally, and using the language of C-K theory, the process of design is considered as a co-evolution of two distinguishable but inter-dependent spaces, the *knowledge* (K) space and the *concept* (C) space. K contains all propositions

[4]Certain forms of what would be called flying ships exist, we just use the historical example of C-K as a means to clarify the concepts related to design theory.

that are either known to be true or false to the designer(s), while C is defined relative to K, and contains all the propositions which are undecidable (they are neither true, nor false during design) in K. As an example, consider the proposition P_1 : "there exist flying objects" and P_2 : "there exist ships", both are true, thus both belong to K, not to C. However, the proposition P_3 : "there exist flying ships" is considered as undecidable in K, thus belonging to C not K, even if P_1 and P_2 have a value in K. Again, it is instructive to compare this definition of design to searchHatchuel & Weil (2009): if the problem is to search for our keys in a house, it is not a design problem because we do not formulate any concept, we know what are keys and we will recognize them when we will find them. However, if we say that we want to build keys that can be found easily, we formulated a concept C_0. The action of formulating a concept (a proposition not in K) is what is called *semantic* disjunction. More precisely, a disjunction happens when we formulate a proposition whose truth cannot be determined with the current knowledge space - it has become disjunct from K. After the initial concept C_0 is formulated, its definition is extended by adding to it a proposition from K, and the effect of adding that proposition is that we add a new property to C_0. The newly added property is not arbitrary, it is usually related to a property of C_0. In other words, suppose "$C_0 = P_1, P_2, \ldots, P_n$" and the chosen proposition is $P' \in K$, and $P_i \in C_0$ one of the properties of C_0, then C_0 becomes "$C = P_1, P_2, \ldots, P'_i, \ldots, P_n$" where $P'_i = P_i, P'$. It is true that it is equivalent to just saying that C_0 becomes "$C = P_1, P_2, \ldots, P_n, P'$", but here we want to stress that P' is related to P_i. If, according to K, P' coupled with P_i are *known* to be part together in the definition of an existing object, then we call C a *restrictive partition*. For instance, if "$C_0 =$ flying, ship", and "$P_i =$ flying", and "$P' =$ wings", then "$C =$ flying with wings, ship" is a restrictive partition of C_0 because we know (from K) that some flying objects have wings (planes and birds). If, however, the chosen P' coupled with P are *not known* to be part together in the definition of an existing object, then we call C an *expansive partition*. For instance, if "$C_0 =$ flying, ship", and "$P_i =$ flying", and "$P' =$ without wings", then "$C =$ flying without wings, ship" is an expansive partition of C because we do not know any flying objects which do not have wings. C-K theory suggests that expansive partitions are the key to innovation and creativity Hatchuel & Weil (2009).

Restrictive partitions can be seen as a way to *restrict* the space of possible objects that can become a solution of the design problem, whereas expansive partitions can be seen as a way to *enlarge* the space of possible objects, because they add unusual properties. Thus, depending on the relationship between the property of the concept (e.g., flying) and the proposition chosen from K (e.g., wings), we can either restrict the space of possible objects or enlarge the space of possi-

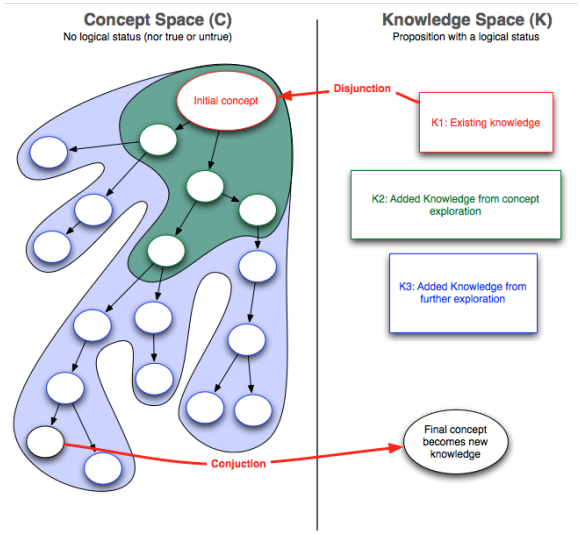


Figure 3.3: An illustration of the design process through of C-K, from Hatchuel & Weil (2009).

ble objects. The process of either building restrictive or expansive partitions can continue, transforming again the concept either by using restrictive or expansive partitions. The process ends once we can prove that a satisfactory concept is true or false, at which point, the concept become known (and thus part of K).

Hatchuel & Weil (2009) represents the concept space as a tree structure. Nodes are concepts. The root node is C_0. Each time we add either a restrictive of expansive partition to a concept (the parent), we add a new node (the child) representing the new concept and link it the parent node. See Figure 3.3 for an example from Hatchuel & Weil (2009).

3.2.5 Formalisms of the K-space

In the design theory literature, different formalisms have been proposed as models of the knowledge space. The original C-K paper used the propositional logic formalism. In propositional logic, the knowledge is a set of logical propositions of the form "there exist ...", e.g., "there exist flying things" that have a truth value. We also call those propositions *attributes*, as they can form an object by conjunction, e.g., "flying and ship". Concepts are conjunctions of attributes of the form "there exist an object with ... and ... and ...", e.g., "there exist an object which is flying and which is a ship", where the attributes are logical propositions from the knowledge set and the conjunction of those attributes form a logical proposition

that is undecidable with respect to the knowledge set. Knowledge expansion is adding new logical propositions with a truth value. A restrictive partition happens by selecting an attribute of a concept (e.g., "flying"), then specializing that attribute (e.g., to "flying with wings") in a *usual* way (we already know flying objects that have wings). An expansive partition happens by selecting an attribute of a concept, (e.g., "flying"), then expanding that attribute in an *unusual* way (e.g., to "flying without helices").

A second formalism based on graphs was proposed by Kazakçı (2007). In the graph-based formalism, the knowledge space is a graph. Nodes are entities, edges are relations between entities. Knowledge is the set of known entities and relationships. Concepts are a set of entities which do not form a connected component. Knowledge expansion means adding a new relationship between two entities or more. A restrictive partition happens by adding an entity to a concept, where the entity is already connected to an entity that belongs to the concept. An expansive partition happens by adding en entity to a concept, where the entity is not connected to any of the entities of the concept. [5]

Table 3.1: Comparison between different C-K formalisms and our proposed *connectionist* formalism

Formalism	**Knowledge**	**Concept**	**Expansive partition**	**Restrictive partition**	**Knowledge expansion**
Propositional logic	Decidable logical proposition	Undecidable logical proposition	Add unusual attribute to concept	Add usual attribute to concept	New logical proposition in knowledge
Graph-based	Known entities and relationships	Set of entities which do not form a connected component	Add entity not connected to any entity of a concept	Add entity which is connected to at least an entity of the concept	Add new entity or relationship to the graph
Connectionist	Weights and data	Unseen combination of features in data	Activate a feature that has never been combined with a subset of features	Activate a feature that has been combined with a subset of features	Update weights with new data

3.2.6 Machine learning as a formalism of knowledge

C-K tells us that expansive partitions happen by exploiting the K space, so a K space with a different nature might lead to different concepts, thus different kinds of designed objects. Although C-K related work has always considered formal and symbolic descriptions of knowledge, in principle nothing prevents the use of a data-driven and connectionist formalism. The first advantage of the connectionist

[5] Several other formalisms such as set theory and forcing, matroids and category theory has been proposed in the literature. Their presentation is out of our current scope and we refer the reader to the cited text above for more information.

formalism is the learning aspect, which makes it possible to implement artificial design agents that learn from data, thus able to extend its K space given there is available data for learning. The second advantage of the connectionist formalism is the use of distributed representations, which are representations that are suitable for learning re-combinable features, which, as we shall argue in later chapters, is an important aspect for generation of novelty.

C-K also tells us that attributes have to be chosen from the K space, but this suggests a mechanism by which those attributes are chosen, and no explicit mechanism is proposed in the C-K literature. As we will see in the following chapters, current generative models can provide a mechanism to select attributes. While it is possible to select attributes without a learning-based model or program, we see learning from data or from experience as a key aspect because, as we will explain, the representation of the knowledge should be driven by data, rather than hard-encoded. In the Table 3.1, we provide a comparison between two formalisms proposed in the C-K literature and our proposed formalism of knowledge, the *connectionist* formalism, which is based on machine learning.

3.2.7 Conclusion

In this section, we have defined design and described C-K theory, a design theory whose main feature is to highlight the interplay between concepts and knowledge. A theory of design can be helpful for giving insights about how to implement systems that can design on their own. Unfortunately, in design theory, design is expected to occur without a specific mechanism. In other words, *processes* that allow the interplay between knowledge and concepts are not clearly described. Furthermore, generation of concepts seems random since it is not specified what governs such a process. If n atomic propositions exist in the knowledge space, the number of C_O that can be formulated is 2^n. It is clear that in practice designers are guided by some selection mechanism. In later chapters, we shall call such a mechanism a value function and demonstrate the effect it can have on novelty generation.

Part II

The relationship between knowledge, representation, and novelty generation

Chapter 4

The effect of representations on novelty generation

In this conceptual chapter, we will first explain in Section 4.1 the links between representation and novelty generation and show that the choice of the representation is crucial in the context of the generation of new objects. In particular, we will show that compressing the representation is not necessarily a good idea to obtain a representation which is good for the generation of new objects. In Section 4.2, through a literature review, we will show that the current way of training generative models in machine learning is related to compression in a way that is not well suited for the generation of new objects. Finally, in Section 5, we will discuss the issue of choosing the value function to guide representation learning and we will define a conceptual framework which we will use to study the implication of using different value functions on the generated objects. This framework relates knowledge, representation learning, and the value function of a designer agent. We will see that the way most current generative models in machine learning are trained is one of the possible instantiations of the proposed framework, and will show other possible instantiations that are less limited from the point of view of the generation of new objects.

4.1 Representations and their effect on the reachability of novelties

In the design theory literature, it is common to describe objects by a set of attributes and their corresponding values, where each attribute represents a component of the object. One example of theories that use attributes to describe objects is the General

Design Theory (GDT) (Yoshikawa, 1981). While this theory has been criticized in Reich (1995), and different other theories have been proposed later, it is one example among many other design theories that used the notion of features and representations to describe existing objects. In the design theories such as GDT, attributes are used to describe existing objects. However, what is usually neglected in that literature is that with those same attributes, it is also possible to synthesize new objects, often much more than the existing ones. For instance, Reich (1995) describes a simple domain of chairs[1] (see Figure 4.1), where the structure of each chair is represented by 10 boolean attributes, where each attribute represent one component of the chair, like "has a seat" and "has a back support" (See Figure 4.2 for details). Using that representation, they define 8 different chairs and explain what the specific chosen structure of each of chair can give as a behavior. The representation, which is chosen by the designer in that specific example, can not only describe existing chairs (the 8 defined chairs), but also be used for designing new chairs. The paper Reich (1995) describes 8 different chairs. However, with 10 boolean attributes, we can obtain a total of $2^{10} = 1024$ different chairs, thus $1024 - 8 = 1016$ *new* chairs. More importantly, and this is a crucial central point of this thesis, *different representations (e.g., different attributes) can lead to a different sets of chairs*. Designers choose not only how to recombine existing attributes but also the abstract representation space in which attributes are described and objects are represented. Quantitatively, a representation with 20 bits lead to much more chairs than a representation with 10 bits, for instance. Qualitatively, different types of attributes can lead to a completely different set of chairs, thus a different expansion in the sense of C-K (see Chapter 3.2), for the same number of bits. The important point we would like to stress here is that finding a *good* representation, in the sense of good for synthesis of new objects, is not neutral: it is *chosen* by the designer and determines what kind of new objects you will be able to generate. It is also not trivial and a key step of the process of design itself.

In order to explain the effect of representations on the generation of new objects, we are going to investigate different situations, where in each situation we will use a generator to build new objects. To illustrate our ideas, we will consider a synthetic image dataset of 16 letters (Figure 4.3), represented as black and white pixels of size 5×9, thus a total of 45 pixels. We denote the synthetic image dataset by K, as it represents *knowledge*, more precisely, the set of *known* objects. In each case, we will consider a different generator that we call G, and that will generate a set of objects R.

[1] It was proposed in the original General Design Theory (GDT) (Yoshikawa, 1981) paper.

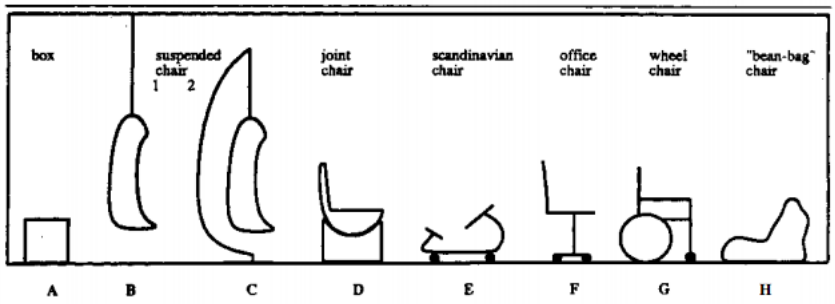


Figure 4.1: Designing chairs in Yoshikawa's example, described in Reich (1995). The figure is from Reich (1995).

		chair							
	structure	A	B	C	D	E	F	G	H
1	*has a seat*	+	+	+	+	—	+	+	—
2	*has a back support*	—	+	+	+	—	+	+	—
3	*has legs*	—	—	+	—	—	+	+	—
4	*has wheels*	—	—	—	—	+	+	+	—
5	*has a vertical rotational dof*	—	+	+	+	—	+	—	—
6	*is lightweight*	+	+	—	—	+	+	+	+
7	*has a hanger*	—	+	+	—	—	—	—	—
8	*has a brake*	—	—	—	—	—	—	+	—

Figure 4.2: Features used to described chairs in Yoshikawa's example. The figure is from Reich (1995).

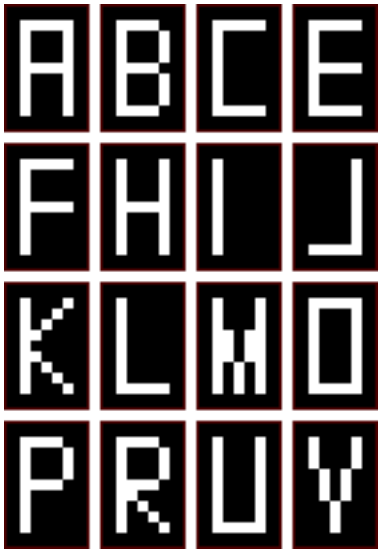


Figure 4.3: Dataset of 16 synthetic letters we used in the experiments.

4.1.1 The dilemma of noise vs novelty

In this first experiment, we will consider a simple generator that operates on the pixel space, and which consists in setting to 1 or 0 each pixel of the 5×9 canvas randomly with probability $\frac{1}{2}$. In Figure 4.4, we show 15 randomly generated images from G. This generator has no knowledge, in the sense that it ignores completely the knowledge set K. Here, the R space is the set of all 2^{45} possible binary images, equiprobable. As we already have a set of 16 letters from K (Figure 4.3), all the rest $2^{45} - 16$ are *new*. However, the vast majority of those $2^{45} - 16$ images have no structure, as we can see from Figure 4.4. Here, we consider the meaning of *structure* as relative to an agent. What is considered "structure" by an agent is what it can recognize or perceive, and what is considered "noise" what it cannot. For instance, an agent which can only recognize letters would certainly consider the vast majority of the $2^{45} - 16$ images as noise. We have seen in this experiment that while the space of objects that can be generated is huge and can lead easily to novelty (almost all the generated objects are new), most of the generated images have no structure. Importantly, in the literature we usually find discussions about generating *new* objects instead of old ones. But in fact, as we show here, it is very easy to generate novelty, almost all objects here are new. What is important is the notion of value that an agent give to objects, which lead to the problem of generating useful novelty ("structure") instead of unuseful novelty ("noise").

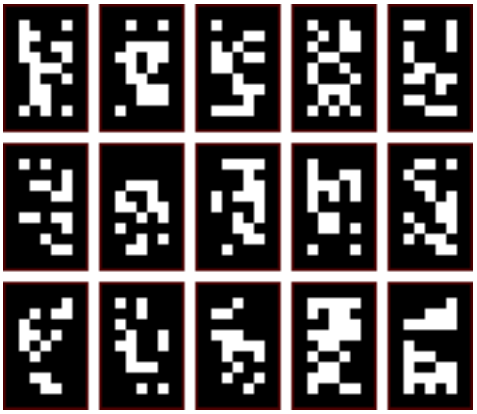


Figure 4.4: 15 randomly generated images from a generator which generates independently each pixel with a global Bernoulli.

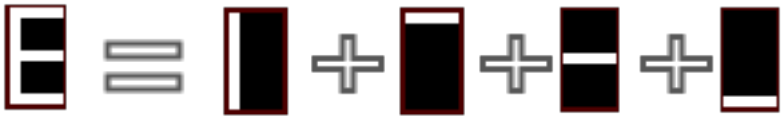


Figure 4.5: The letter *E* can be built by super-imposing four different strokes, three horizontal bars (top, middle, and bottom) and one vertical bar (left).

4.1.2 The effect of having a good representation

In this second experiment, we will re-represent the 16 synthethic letters, originally in pixels, using a set of attributes, just like in the chairs example (Figure 4.2). Each attribute will represent the presence or the absence of a kind of stroke. For instance, the letter *E* can be represented using four strokes, three horizontal bars (top, middle, and bottom) and one vertical bar (left), as we can see in Figure 4.5. We do a similar decomposition for all the remaining 15 letters. In Figure 4.6, we show the 10 proposed kinds of strokes we use to represent the letters. In Table 4.1, we show the representation of each letter in the stroke feature space. Given a letter represented in the stroke feature space, it is possible to "build" the letter in the pixel space by super-imposing the strokes that are active (that is, the strokes that are equal to 1, see Table 4.1).

The representation we just described can not only describe existing objects (the 16 letters) but also be used to generate new objects. In this experiment, we will consider a simple generator G that operates on the stroke space, rather than on the pixel space. To generate an object, we first select randomly whether each stroke is activated or not (1 or 0), according to the ratio by which the stroke is

Figure 4.6: The 10 features that are used in Table 4.1. From the left to the right of the figure: left vertical bar, right vertical bar, top horizontal bar, middle horizontal bar, bottom horizontal bar, top belly, bottom belly, top diagonal bar, bottom diagonal bar, double diagonal bar.

Table 4.1: Representation of each letter using the stroke features, which we call y.

Letters	A	B	C	E	F	H	I	J	K	L	M	O	P	R	U	V
Features																
left vertical bar	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0
right vertical bar	1	0	0	0	0	1	0	1	0	0	1	1	0	0	1	0
top horizontal bar	1	0	1	1	1	0	0	0	0	0	0	1	0	0	0	0
middle horizontal bar	1	0	0	1	1	1	0	0	0	0	0	0	0	0	0	0
bottom horizontal bar	0	0	1	1	0	0	0	1	0	1	0	1	0	0	1	0
top belly	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0
bottom belly	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
top diagonal bar	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
bottom diagonal bar	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
double diagonal bar	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1

present in the knowledge set K, on the 16 letters. Once the 10 strokes activations are selected, we build the image in the pixel space by super-imposing the strokes that are active (equal to 1, see Table 4.1). Here, the set of possible images that can be generated is $2^{10} = 1024$, which is much less than in the case of Section 4.1.1(2^{45}). In Figure 4.7, we show 15 generated images from G. As we can see, contrary to Section 4.1.1, the generated images are much more structured, now that we changed the representation on which we do the generation. Importantly, although the generated images (Figure 4.7) look more structured than the ones in the Section 4.1.1 (Figure 4.4) due to changing the representation, we have not attempted to show whether the chosen representation is *good* in any formal sense, but if it is, there is a very tiny probability $p = \frac{1024}{2^{45}} = 2^{-35}$ of generating the 1024 images that can be generated from G, using the generator that operates on the pixel space that we used in Section 4.1.1. We have shown in this second experiment that by exploiting knowledge from data and changing the representation on which we do the generation, we can indeed help generation. In other words, we have shown that by changing the representation, the kind of expansions (see Chapter 3.2) that we

Figure 4.7: 15 randomly generated images from a generator which generates using the stroke features (see Table 4.1 and Figure 4.6), rather than the pixel space.

can reach change considerably.

4.2 Compressing the representation can kill novelty

We have previously demonstrated that re-representation of objects is important, through the use of the stroke features (which we define in Table 4.1) instead of the raw pixel space. In machine learning, the question of finding a good representation (for e.g., supervised learning) is crucial and a large fraction of the literature is dedicated to this question. However, the question of how we could find a good representation *for* design, or generation of new objects is not studied. In machine learning, the literature of generative modeling addresses the question of how to train good generative models to generate objects from the same distribution of the training data. In generative modeling, traditionally, training data is associated with an unknown generative probability distribution from which the data is sampled independently and identically. The goal of learning is to characterize or specify this distribution. The mechanism of generation is then to sample from the distribution randomly (when a constructive sampler can be defined), or frame the problem as an optimization where new objects are generated in other ways and scored by an explicit probability density.

In generative modeling, as we argue in Section 4.2 through a literature review, current generative models mostly rely on learning a representation that optimizes a compression objective function. In our letters example, the stroke feature space re-represent each image using 10 bits, but we only have 16 examples. In the following experiments, we are going to re-represent the letters using much fewer bits

in order to achieve compression. We will argue that compression eliminates the space needed for design, and so the main reason current techniques generate any interesting novelty is that they fail at perfectly achieving their compression goal.

4.2.1 Maximal compression leads to generation of known objects

In this third (thought) experiment, we re-represent each example with only 4 bits, as we have 16 examples. The set of possible images in R is $2^4 = 16$. We re-represent each letter using a boolean "indicator" vector of 4 bits. The association between images and 4 bits codes can be chosen in an arbitrary way, with the constraint that each code will map to exactly one of the letters. By construction, we will not be able to generate any new images because all the codes map to one of the known letters. This thought experiment shows that the strict compression objective function is harmful if our goal is to generate new images, as it will leave no place in the representation for other objects than the ones that are given.

4.2.2 Implicit Compression through regularization also leads to generation of known objects

Next, to enlarge the space of possibilities, we are going to increase the capacity of the representation. As we saw in the previous experiment, to achieve maximal compression we need to represent each example with 4 bits. With less than 4 bits, we will not be able to distinguish anymore between the 16 examples of the training set. What happens with more than 4 bits? In this experiment we use a binary representation with 5 bits of capacity. That is, we re-represent each letter, using a binary representation of 5 bits. In this fourth experiment, the stroke space, which has 10 bits, will be the "input" space, and we will call it y. We will *learn* to re-represent each letter using a feature space of 5 bits, which we call z. In order to represent the 16 letters of the training set with 5 bits, we randomly assign each of the 16 examples to 16 distinct vectors in the compressed space, leaving $2^5 - 16 = 16$ unused points on the compressed space. Then, we optimize a non-linear (supervised) mapping g from z (5 bits) to y (10 bits)[2], such that, $g(z) \approx y$ for all the existing 16 letters, where g is the non-linear mapping. The mapping g can be seen as the decoder of an autoencoder, where the input has 10 bits (the stroke space), the code is z (the compressed space), and the encoder is given and fixed (the encoder is just mapping from y to z using the 16 distinct vectors chosen randomly). The function g will allow us to give the y corresponding to 16 the unused points of the compressed space z. Also, as commonly used in machine learning, we will apply a regularization criterion to the non-linear mapping, to

[2]We found that a linear mapping was not sufficient to learn that mapping.

achieve compression. The loss function for optimizing the mapping $y = g(z)$ is the mean squared error with a regularization criterion weighted by a coefficient. We use a neural network with two hidden layers for the mapping g[3] and we use an $L1$ loss on the second layer as a regularization, which has a sparsification effect (the activations of the second layer are encouraged to be equal to zero). We search for the biggest regularization coefficient that still allow the objects of the training set y to be correctly predicted from z using g. We train two identical models, one with regularization and one without. Both models achieve a perfect prediction accuracy of y from z, for the letters.[4] In Figure 4.8, we show the images corresponding to the 16 unused points of the compressed space, with and without regularization. The images shown in Figure 4.8 are built as the following. First, we use the learned mapping g on the 16 unused codes of the compressed space z to transform them to the stroke space y. Then, we build the images by super-imposing the features that are predicted as active by g on the stroke space. We can see that with regularization, all the unused points correspond to examples from the training set, while without regularization, some correspond to examples outside the training set. The point of this experiment was to show that regularization techniques like the ones used in generative models of machine learning are harmful if the goal is to generate novelty, because the effect is that even unseen points during training in the feature space are mapped to "known" points, that is, points from the training set.

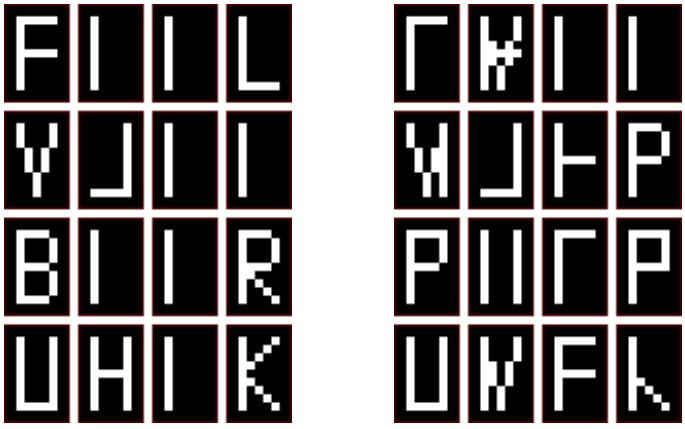
4.2.3 Overfitting and generalization are orthogonal to novelty generation

In machine learning, rather than "compressing" only the training set, what is usually sought is to compress well unseen data from the same distribution of the training set as well as the training set. That is, for a probabilistic model, we do not want the model to put all the probability mass in the training data only, we also want it to put some probability mass on (unseen) test data ("generalize"). In our synthetic dataset of letters, we used all the available 16 examples without considering the question of overfitting. However, consider the asymptotic regime where we are able to sample as many examples as we want from the assumed true distribution. Suppose our objects are represented as a vector with discrete values[5]. Then, the set that contains all possible objects from the assumed true distribution is usually very big but countable and finite. For instance, suppose we are able to sample from

[3] As y is binary, we apply a thresholding operation to the output of g to have binary outputs

[4] Since the goal is not generalizing, rather showing the effect of regularization on the design space, overfitting is not an issue.

[5] This is not an unrealistic assumption, since in real situations we are dealing with digital computers, where all objects have to be discretized.



(a) With regularization (b) Without regularization

Figure 4.8: By regularizing the generative mapping, all the generated examples are identical to examples from the training set, while if we remove regularization, some new examples are generated. See the text for more details.

the distribution of handwritten digits, as many times as we want. Eventually, if we sample a large number of times from the assumed true distribution, we will be able to cover the full (countable and finite) set of possible digits. In this case, there is no problem of overfitting, because the goal is just to be able to memorize the full set of all possible objects with a model, for instance. Still, the goal of design and novelty generation is to go beyond that set, the set of of all possible digits. Thus the question of overfitting in the machine learning sense is not the main issue in the context of design. In our setup where we use a dataset of synthetic letters, the 16 examples should rather be considered as representing the set of "all possible objects" given to the agent in a simplified world.

4.2.4 Current deep generative models are trained to compress

As we illustrated in the previous examples, compressing the representation is not necessarily a good choice for learning representations which are good for generating new objects. Here, through a literature review, we will explain more concretely why current objective functions used in generative models have a compression effect, and thus do not lead to a representation which is suitable for the generation of new objects. We will relate the field of data compression, a branch of information theory, to the current paradigm used to train generative models in machine learning.

A brief review of coding theory

Coding theory is a broad field which studies *codes* and their use in different applications. Among the applications where codes are studied is data compression, for which the goal is to find efficient codes in order to store (e.g., in a hard disk) or send (e.g., through the network) data, where the efficiency is measured in terms of the number of bits to store or to send (size of the code) such that the original data can be recovered from the code. Data to compress can be any binary string of bits such as images, audio, and text files. Codes are binary strings of bits as well, which we would like to be smaller than the original data. In data compression, we design a tuple of functions (f, g) where f is called the encoder and g the decoder. The encoder f takes an object $x \in D$ (e.g., an image) and returns the code as a binary string, $f : D \rightarrow \{0, 1\}^*$. The decoder g takes a code and reconstructs the original object corresponding to the code, $g : \{0, 1\}* \rightarrow D$.

The theory of data compression is studied within information theory, a field which was initiated by Claude Shannon. In information theory (Shannon, 1948), we assume our objects are sampled from a fixed probability distribution, and our goal is to design f and g such that the expected code length is the smallest possible, while still being able to recover the original objects. In the simplest case, a set of objects are sampled i.i.d[6] from a discrete probability distribution $P(X = x), x \in D$. The expected code length can be defined as: $\bar{L} = \sum_{x \in D} P(X = x) l(f(x))$, where $l(f(x))$ is the length of the code corresponding to x. Note that different objects can be mapped to codes of different length.

In data compression, we differentiate between lossless and lossy compression. In lossless compression (e.g., PNG, ZIP), the goal is to design f and g such that the expected length is minimized and the objects are *exactly* recovered, that is, $\forall x \in D, g(f(x)) = x$. Moreover, we add a constraint that such codes are uniquely decodable (Grunwald & Vitányi, 2004), that is, when we encode a concatenated binary string of several objects $x_1, x_2, \ldots, x_N$ as $f(x_1)f(x_2)\ldots f(x_N)$, we are able to reconstruct the original set of objects without leading to ambiguities. Source coding (Shannon, 1948), created by Claude Shannon, is a branch of information theory which studies lossless data compression, and establishes a fundamental limit on data compression, which is that $\forall f, g, \bar{L} \geq H(X)$ (Shannon, 1948; Grunwald & Vitányi, 2004), where $H(X) = \sum_{x \in D} P(X = x) \log_2 \frac{1}{P(X=x)}$ is called the entropy of X. Moreover, the bound is achievable, and it is possible to approach the entropy lower bound with an arbitrary precision, that is, $\forall \epsilon > 0, \exists f, g, \bar{L}(f, g) \geq H(P) + \epsilon$ (Wiegand et al., 2011). Concretely, source coding theory tells us that for an efficient coding we should use codes of varying length and that frequent objects should be assigned to codes with small length, while rare objects should be

[6] Identically and independently distributed

assigned to codes with larger length. It is not possible to use small codes for all objects, because as the number of objects increase, the number of possible codes with small length will not be enough to represent all objects. There is a close relationship between density estimation and compression. In source coding, the probability distribution of objects $P(X)$ is supposed to be known and given, but in real setups it is usually not the case. In machine learning, unsupervised learning is frequently formulated as a density estimation objective. Density estimation can be used to estimate the probability distribution of objects from a sample, and thus can be helpful for compression, because it can give us the number of bits to assign per object, which according to the entropy should be $l(x) = log_2 \frac{1}{P(X=x)}$.

If we can afford to allow errors in the reconstructed objects, we call the task lossy compression (e.g, JPEG, MPEG). In lossy compression, we are given a distortion function $d(x, \hat{x})$, which measures the amount of error between the original x and the reconstructed object $\hat{x}$, where $\hat{x} = g(f(x))$. The distortion function depends on the domain. For instance, in lossy image compression, we usually use a perceptual distance (Wang et al., 2002) which is meant to take into account which details are important for a human and which details are not when comparing between the original and the reconstructed object. The *rate-distortion theory* (Shannon, 1959; Wang et al., 2002), also created by Claude Shannon, studies lossy data compression, and like source coding, establishes fundamental limits on the achievable bounds of lossy compression. In the rate-distortion theory, there is a trade-off between the amount of distortion and the amount of compression. In other words, to achieve a better compression, we are obliged to lose some details of the original objects, as measured by the distortion function. Lossy data compression is formalized as a constrained optimization problem.

First, we set a maximum allowable expected distortion $\mathbb{E}[d(x, \hat{x})] \leq D_0$, where the expectation is taken over $P(X)$. Then, among all (f, g) that achieve an acceptable expected distortion (smaller than D_0), we search for the ones that achieve the best amount of compression, measured by $\bar{L} = \sum_{x \in D} P(X = x) l(f(x))$. The rate-distortion theory establishes a fundamental lower bound on the amount of compression given a maximal distortion through the information rate-distortion function, which can be written as $R(D_0) = \min_{(f,g), \mathbb{E}[d(x,\hat{x})] \leq D_0} [I(X; \hat{X})]$ (Wiegand et al., 2011), where $\hat{X}$ is a random variable obtained by reconstructing X (the original random variable), $\hat{X} = g(f(X))$, $I(X; \hat{X})$ is called the mutual information between X and $\hat{X}$, and $R(D_0)$ is called the rate. Hence, just like in lossless compression where we cannot beat $H(X)$, in lossy compression given a maximal expected distortion D_0 we cannot beat $R(D_0)$. The mutual information between X and $\hat{X}$ can be written as $I(X; \hat{X}) = H(X) - H(X|\hat{X}) = H(\hat{X}) - H(\hat{X}|X)$. $H(X)$ can be interpreted as the amount of uncertainty the "receiver" has about X

or, in other words, the number of bits required to send to the receiver so that it recovers the original object. $H(X|\hat{X})$ is called the conditional entropy, and can be interpreted as the amount of uncertainty the receiver has about X after seeing $\hat{X}$, or in another words, the number of missing bits to specify X when $\hat{X}$ is seen by the receiver. The setup of the rate-distortion information function is general and considers random mappings as well as deterministic mappings from X to $\hat{X}$. For simplicity, we consider f and g to be deterministic. If g is deterministic and bijective, we have $I(X;\hat{X}) = I(X;Y)$ where $Y = f(X)$ is the code of X, and $\hat{X} = g(Y)$. By definition, $I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$. We also have $H(Y|X) = 0$ because f is deterministic, thus we can write $H(X) = H(Y) + H(X|Y)$, as a decomposition of $H(X)$. In lossless coding, we cannot beat $H(X)$. In lossy coding, we can do better than $H(X)$ by transmitting only $I(X;Y) = H(Y)$ and allowing the remaining $H(X|Y)$ to be lost. In this case, the goal of lossy coding is to minimize the entropy of the code $H(Y)$ subject to the constraint on the distortion. One can also write the converse of $R(D_0)$, $D(R_0)$ where a maximum allowable rate is fixed and we find the minimum amount of distortion. It can be shown that $R(D_0)$ and $D(R_0)$ are inverse of each other, thus the two problems are equivalent.

Deep generative models are trained to compress

Vector quantization algorithms (Wiegand et al., 2011) (e.g., K-means) attempt to find a minimal distortion given a maximum rate to not exceed. In vector quantization, the codes have a fixed length and the length of the codes is related the total number of centroids k. More concretely, each point is represented as the centroid which is the closest to it (the f function) and the index of the centroid is transmitted or stored. To encode k centroids, $\log_2(k)$ bits are needed per point. Once the index is received, the value of the vector of the centroid corresponding to the index is retrieved (the g function). In vector quantization, the trade-off between the amount of compression and the amount of distortion is modulated by the number of centroids. As the number of centroids increase, more bits per point are needed, but we achieve a better distortion. If the number of centroids is small, a small number of bits are needed but we have an increased distortion.

Regularized autoencoders can also be related to the rate-distortion problem. In regularizated autoencoders, we usually optimize a sum of two terms, a reconstruction and a regularization term. The reconstruction term corresponds to the distortion, while the regularization term can be seen to be related to the desired amount of compression. Thus, mixing the two corresponds to a trade-off between distortion and compression. In variational autoencoders (Kingma & Welling, 2013), Zhang et al. (2017) shows that the regularization term used is an upper bound of $I(X;Y)$

where $Y = f(X)$, which by the data processing inequality (Beaudry & Renner, 2011) is greater than $I(X;\hat{X})$, $I(X;Y) \geq I(X;\hat{X})$ where $\hat{X} = g(Y)$. Thus, if the regularization term is minimized, so is $I(X;\hat{X})$ which measures the amount of compression according to the rate-distortion theory. In variational autoencoders (VAEs), both f and g are stochastic and the regularization term in VAE forces the encoder to map all points to a spherical Gaussian, that is, the regularization term wants $\forall x, P(Y|X = x) = \mathbb{N}(0, \mathbb{I})$. The regularization term can be seen as a enforcing the mapping f to be contractive, as all the points are forced to map to a spherical Gaussian centered at zero. Minimizing mutual information and forcing the map to be contractive are related. When the mapping $Y = f(X)$ is contractive, it means very different points $x \sim P(X)$ will be mapped to the same code $y \sim Y|X$, which, in an information theoretic perspective, means that there will be still a lot of uncertainty about X after the receiver sees Y, implying a small $I(X;Y)$ and thus a small $I(X;\hat{X})$ as well because $I(X;Y) \geq I(X;\hat{X})$, and thus a good compression (but a bad distortion). In contractive autoencoders (Rifai et al., 2011), both f and g are deterministic, and the f is encouraged to be contractive by penalizing the derivatives of it with respect to the input. Denoising autoencoders (Vincent et al., 2008) have been shown to be related to contraction by (Alain & Bengio, 2014), where the derivative of the reconstruction function $g \circ f$ with respect the to the inputs are penalized instead of the derivative of f in contractive autoencoders. (Rifai et al., 2011) states that sparse autoencoders as well are likely to be contractive, as the majority of the components of the code are zeroed out.

The formulation of Generative Adversarial Networks(GANs) in Goodfellow et al. (2014) optimize the The Jensen–Shannon divergence[7]. While maximum likelihood is equivalent to minimization of $D_{\mathrm{KL}}(q \parallel p)$, where p is the assumed true distribution and q is a surrogate distribution, the Jensen-Shannon divergence is an alternative, and is defined as $\mathrm{JSD}(p,q) = \frac{1}{2}D_{\mathrm{KL}}(p \parallel m) + \frac{1}{2}D_{\mathrm{KL}}(p \parallel m)$, where $m = \frac{1}{2}(p+q)$. As pointed out by Theis et al. (2015), when the true distribution is multi-modal, likelihood tends to focus more on giving mass to points from all modes, with the risk of giving mass to points between the modes ("spurious" points). On the other hand, JSD tend to focus on fitting one or few modes correctly, with the risk of ignoring the other modes, an issue called "mode dropping".

As we have seen, there is a link between compression and the way the current machine learning models in generative modeling are trained. In particular, the regularizations used in autoencoders have a contraction effect on the code around the points in training data, and thus push all the points outside the data distribution

[7]Different formulations have been proposed in the literature further which do not optimize the JSD anymore, but it is out of the scope of this thesis.

to map to points of the data distribution. However, as we explain in Section 4.1, this has a detrimental effect if our goal is to learn a representation that will be useful for the generation of new objects. One might object that those kind of models that compress very well the training data do not "generalize" well, in the sense of not putting any mass on on unseen data from the same distribution. Our answer is that even if the model can compress well unseen data from the same distribution of the training data, the learned representation will it still not be able to lead to the generation of new types of objects. That is, if the trained model is able compress well the distribution of handwritten digits (training and test data), one can use the learned representation to generate variations of digits, but one cannot used the same representation to generate new types of characters, for instance letters.

Chapter 5

Value functions beyond compression

We have established in the previous chapter the importance of choosing a *good* representation for novelty generation, and that objective functions currently used in machine learning, such as the likelihood, aims to suppress the generation of new objects. What might be a learning objective beyond the likelihood criteria? [1] In order to explain the effect of using other value functions on what can be possibly learned and generated, we will present the KRV (Knowledge-Representation-Value) framework, a conceptual framework clarifying the relationship between knowledge, representation, and value in the context of the generation of new objects.

Let us consider an agent A that seeks to create novelty. Let us consider a specific target domain D from which the agent would like to generate objects. We shall assume that each object in D is characterized by its *raw* representation and that the agent is allowed to re-represent the objects using an intermediate and different representation than the raw representation. When the agent is asked to generate an object, the output will be on the raw representation space. For instance, in the context of synthetic image generation, the raw representation can be considered as pixel intensities. In music, it can be considered as sound waves. For building a sculpture, it can be the outcome (the state of the environment) of a sequence of actions among the possible actions that the robot controller can perform on some material. Suppose for simplicity that D, the set of all possible objects in the target domain, is countable but not necessarily finite.

We further assume that agent A can learn and that it has a state of knowledge

[1] Note that the terms value function and objective function (which is more used in machine learning literature) are used interchangeably and should be considered synonyms.

that can evolve through. We consider that the state of knowledge of *A* at some instant t as the set *K* of all the objects the agent can already build, thus it is represented in the raw representation space. Suppose $K = \{x_1, x_2, \dots x_N\} \subset D$. Now, suppose that the agent *A*, based on *K*, just like in the chair example of Section 4.1, builds an intermediate representation $h \in H$ and a mapping from the intermediate representation to the raw representation that allows it to generate objects, that is, $f : H \to D, x = f(h), h \in H, x \in D$. We call the set of objects that the agent can represent, and thus can potentially generate, as the set $R = \{f(h), h \in H\} \subset D$. Finally, suppose also that there exists a value function $v(x), x \in D$, that is, a mapping from D to real numbers, which is either known by the agent (external reward) or currently unknown, which is typical in a design task. When the value function is unknown, the agent has either to guess it a priori or use some internal (intrinsic) value function. The value function defines a set of *valuable* objects, which are objects for which the value is higher than some threshold θ, and which we call the set $V \subset D$, where $V = \{x \in D, v(x) > \theta\}$ for some $\theta \in \mathbb{R}$. We consider that the triplet $(K_t, R_t, V_t) \subset D \times D \times D$ defines the state of the designer agent at some instant t, and we consider in the general case that K_t, R_t, and V_t all have intersections with each other.

While we do consider that the state of the agent can evolve over time (any component of the triplet can be changed over time), we shall not need to describe the dynamics of the agent state (K_t, R_t, V_t). Thus, we just denote the state of the agent as (K, R, V). We illustrate the agent state (K, R, V) in Figure 5.1. Given the agent state (K, R, V), we can also define the following sets. $D \setminus K$ are novel objects relative to the current knowledge K of the designer agent. However, not all of them are valuable. $V \setminus K$ are novel and valuable objects, which is what we consider that a designer agent would like to find. Unfortunately, not all of them can be represented and generated by the agent, thus there exist some valuable objects that are unreachable to the agent. $(V \cap R) \setminus K$ are all the novel, valuable, and objects that can be generated by the designer agent, while $[V \cap (D \setminus R)] \setminus K$ are the valuable and novel ones which the agent cannot reach due to its limited representation space. We consider that V is not necessarily a superset of K. Under this definition, $D \setminus V$ is what we would call "noise". Note that if V are letters, digits are noise - which goes beyond the traditional conception of noise. In the following, we will instantiate the KRV framework in different situations, relate them to novelty generation, and compare them.

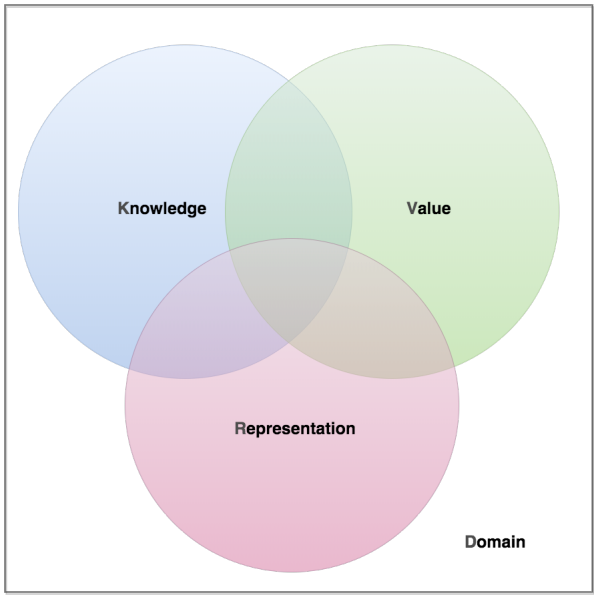


Figure 5.1: The components of the KRV (Knowledge-Representation-Value) framework.

5.1 Probability as a value function

In machine learning, and more specifically in generative modeling, we have a training set $\{x_1, x_2, \ldots, x_N\} \subset D$, which can be considered as the knowledge set K. It is usually assumed in machine learning that the training examples are sampled randomly from a true but unknown probability distribution p_{true}. In that case, the value function $v(x)$ can be considered as the probability density $v(x) = p_{\text{true}}(x)$. Thus, the V set is the set of likely points under the true probability distribution.

Most generative models in machine learning are trained to model an assumed true distribution that would have generated the knowledge set K, using maximum likelihood criteria for instance. The goal in that case is to estimate the parameters of the generative model, such that all $x \in K$ as well as unseen likely points from the true probability distribution are also likely under the generative model we train. In generative modeling, V (e.g., the set of all possible digits) is a superset of K (e.g., the set of training digits), and we would like to align R, which is what we actually end up generating after training the model, with V. Thus we want all the likely points under the generative model to be the same than the ones of the assumed true probability distribution, such that when sampling randomly from the generative model we end up generating likely objects under the true distribution.

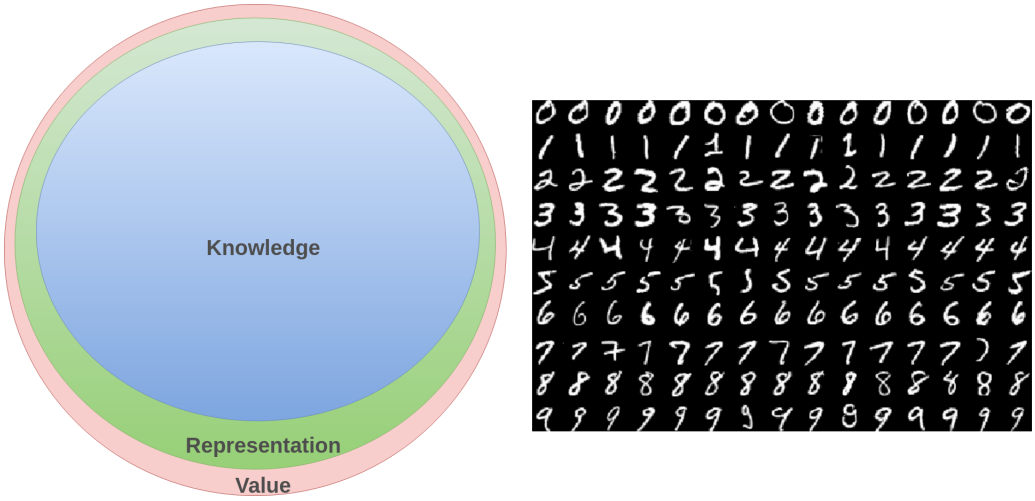


Figure 5.2: The components of the KRV framework when training with maximum likelihood or an alternative to it to model the distribution of the data. In ML, K is the training set, a subset of handwritten digits in this example, and the value set contains are all the likely points according to the assumed true but unknown probability distribution, which in this example is the probability distribution of handwritten digits. V is a superset of K and the goal in machine learning is to make R as indistinguishable as possible from V.

See Figure 5.2 and Figure 5.3 for an illustration.

5.2 Driving the representation with an external value function

5.2.1 Without using knowledge

In this setup, we are given an external value function $v(x)$ that judges how much the objects generated by the agent are valuable. However, the agent does not use knowledge and the value function to guide the representation of the objects. The agent uses the raw representation directly to generate objects, then it checks whether they are valuable by using $v(x)$. In this setup, the objects that can potentially be generated by the agent are all the possible objects from the domain D, and a tiny fraction of those objects that can be generated are valuable. See Figure 5.4 for an illustration.

(a) K (b) V (c) $R \cap V$

Figure 5.3: Examples of K (knowledge set), V (value set) and $R \cap V$ (generated by the agent and valuable) when training with the generative model with maximum likelihood or some proxy of it.

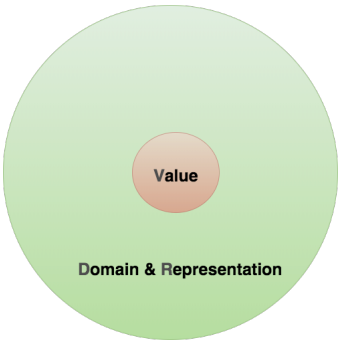


Figure 5.4: The components of the KRV (Knowledge-Representation-Value) when the knowledge and the value function are not used to guide the representation. The objects are generated directly in the raw representation and valuable objects are selected among them. The objects that can be potentially generated are all possible objects from the domain, thus $R = D$, and the set of valuable objects are a tiny fraction of $R = D$, as D is usually very big (e.g., even the domain of black and white images of size 5×9 contain a total of 2^{45} objects).

5.2.2 Using knowledge

In this setup, we impose an external value function (external to the agent) that judges how much the objects generated by the agent are valuable. The agent is given a knowledge set K, a value function $v(x)$ and is asked to generate new objects, then it is judged by its capacity to generate new and valuable objects. The value function $v(x)$ is given to the agent, in the sense that a symbolic formula of it is available. The agent is allowed to exploit the value function (e.g., compute its derivatives) during representation learning and/or generation. This setup has been instantiated in one of the experiments of Nguyen et al. (2016a), where they use a generative model trained to generate images from a set of classes to exploit a classifier trained on another set of classes (these classes are new and have never been seen by the generative model) in order to generate images from the new set of classes.

5.3 Exploiting semantic labels

In this setup, an external agent (e.g., a human) "labels" each example from K with a set of M semantic attributes $y_1, y_2, \ldots y_M$. The agent A is only asked to learn a mapping from the semantic attributes to the raw representation of objects. The mapping will allow the agent to generate objects that correspond to combinations of the semantic attributes which are outside the training set. The set V is considered to be all possible combinations of the semantic attributes. If the semantic attributes are binary and independent, the maximum number of different objects that can be generated are 2^M, where M is the number of semantic attributes. Like in the setup of Section 5.1, V is a superset of K, because the training set K is a set of objects that corresponds to combinations of semantic attributes. This setup corresponds to the use of conditional generative models in the generative modeling literature (e.g., see Jin et al. (2017)), where a set of semantic attributes (*"tags"* rather than just one class/category) per object are given in order to exploit the exponential power of recombination. This setup allows to generate *hybrids* by activating semantic attributes that have never been seen *together* in K. For instance, if the semantic labels correspond to categories, e.g. each image in K is labeled as dog or cat or fish, it is possible to generate hybrids such as dog-cats, or cat-fishes. Hybridation have been experimented in Nguyen et al. (2016a). An example of hybridation is given in Figure 5.5.

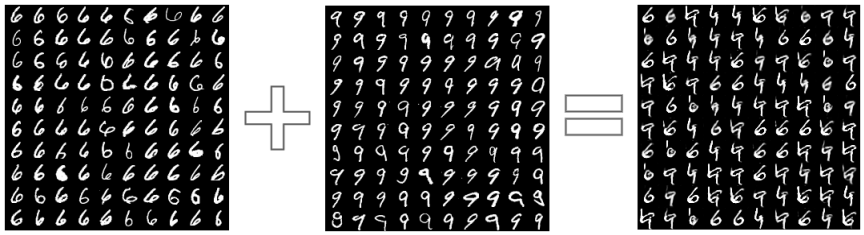


Figure 5.5: Illustration of hybridation of known classes using Generative Adversarial Networks(GANs) (Goodfellow et al., 2014). The two first grid of images refer to known classes (class "6" and class "9"), thus they belong to K. The third grid of images is the result of hybridation (hybrids of "6" and "9"), and represents $R \cap V$. In this example, we train a GAN (Goodfellow et al., 2014) to do class hybridation. The discriminator receives either real samples (from the class 6 or the class 9) or samples from the generator. The first goal of the discriminator is to predict whether an image is real or fake. The second goal of the discriminator is to predict the right class, that is whether the image corresponds to a 6 or to a 9. The generator tries to fool the discriminator in two ways. First, it tries to make the discriminator predict that the fake images are real. Second, it tries to make the discriminator incapable of deciding whether a fake image is 6 or a 9, by maximizing the entropy of the posterior probability of the classes.

5.4 Constructing internal value functions

5.4.1 Without external evaluation

In this setup, the agent is builds its own value function based on on the knowledge set K. As the agent builds its own value function based on K, we consider that the V and the R sets are exactly the same because we assume the agent generative mechanism relies on its own internal value function, thus only generating objects that are valuable according to the value function. However, $R = V$ is not necessarily the same than K. In other words, the agent is able to generate new objects, outside of K, using its own value function. One possible implementation example of this kind of agents would be to make the agent learn a high-level representation h of the objects from K in terms of a set of *features* and use the same representation to generate new objects by recombining the learned features in a new way. See Figure 5.6 and Figure 5.7 for an illustration. A high-level representation means that it should only contain a coarse level of details, leaving the rest of the details to the generative mapping $x = f(h), h \in H, x \in R$. A new recombination of the high-level features means a combination that is not seen in the knowledge set K. While it is not clear what a new recombination means when the representation h is continuous, if it is discrete (e.g., binary) it is simple to distinguish between old and new objects in the representation space, because the set of all possible objects that can be generated R is countable and finite.

The agent is able to represent objects from K using a high-level representation, combined with a generative mapping f that fills the low-level details. The agent can define the valuable objects as all the objects that use the same low-level features than objects in K, that is, the set of all objects that use the generative mapping f to fills the details from the high-level representation. A subset of those objects would necessarily be the objects from the set K. However, the new and valuable objects could be all the objects that still use the generative mapping f, preserving the low-level features that are used in K, but combines the high-level features in a novel way, not seen in the K set. We call those new and valuable objects (according to the agent) the *structural holes*.

In the digits example, an example of high-level features that could be learned are strokes. In that setup, the agent can learn to represent each image as a set strokes (we have different "types" of strokes) and their position in the image. The high-level representation would only contain a description of which kind of strokes are used and their position, while the generative mapping f would "fill" the details by "drawing" the strokes. Then, the agent could reuse the same learned strokes to generate different kinds of symbols than digits. We explain this setup in Chapter 7 more in detail.

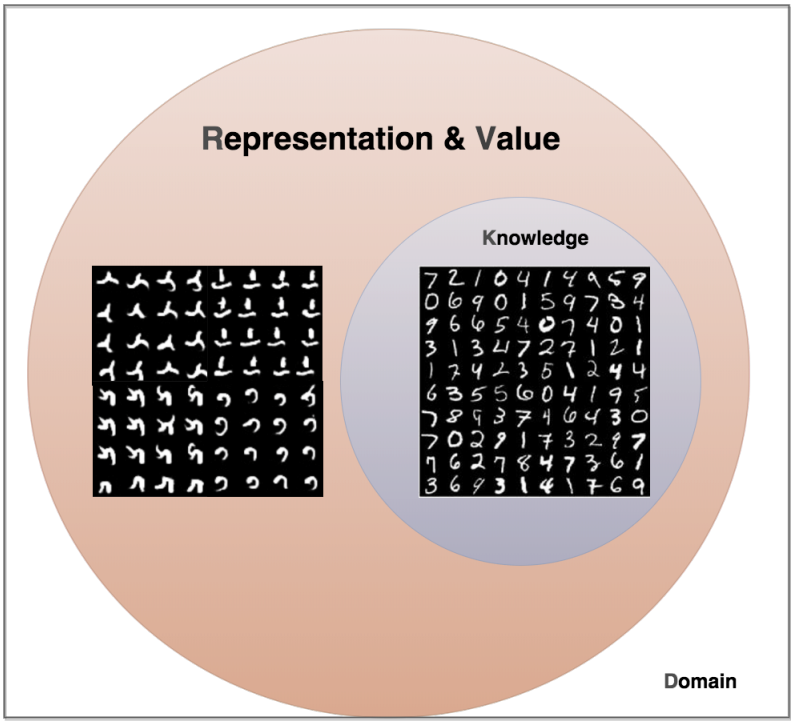
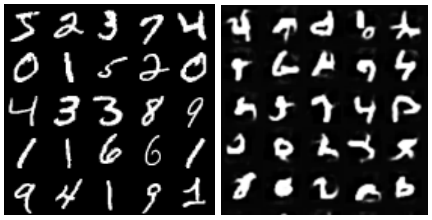


Figure 5.6: The components of the KRV framework when the value function is internal and learned from a knowledge set. In this setup, as the value function is designed by the agent itself, $R = V$, but they are not necessarily equal to K. The images generated at the left are from Chapter 7, where we generate using a sparse convolutional autoencoder trained on MNIST.

Another way to implement an agent which has a learnable internal value function would be the theory of curiosity which was proposed by Schmidhuber in Schmidhuber (2010), we describe it in Section 3.1.2.

5.4.2 With external evaluation a posteriori

This setup is like the one of Section 5.2, except that the external value function is not given to the agent during representation learning and generation. The agent is asked to generate new objects, and it is judged in *the future* whether it can generate new and valuable objects. This resembles more what design looks like in real situations, contrary to Section 5.2 where the value function is given to the agent and the agent is allowed to exploit it during learning and/or during generation. As an example, the knowledge set could be from a domain, e.g. handwritten digits,



(a) K (b) $R \cap V$

Figure 5.7: The components of the KRV framework when the value function is learned from the knowledge set (training set). In this setup, as the value function is designed by the agent itself, $R = V$, and thus $R \cap V = V = R$, but they are not necessarily equal to K. The images generated in (b) are from Chapter 7, where we generate using a sparse convolutional autoencoder trained on MNIST.

and the value function will look for objects from another domain, e.g., handwritten letters. See Figure 5.8 and Figure 5.9 for an illustration. The value function $v(x)$ can be defined for instance using an external agent B that predicts whether x is a letter, $v(x) = \text{is\_letter}(x)$. In Chapter 8, we propose a definition of novelty generation as the generation from new classes unseen to the agent and use this setup as a task.

5.5 The general designer

This is a general setup of a designer agent, which, after receiving feedback from the external world, updates its knowledge set K and re-learns the representation to be able to generate more valuable objects. This is the most general designer agent and it is adaptive in two different aspects. First, the knowledge set K is not fixed and is updated over time. Second, it does not need to have a value function a *priori* and does not consider that the value function is fixed, that is, the value function $v(x)$ can change over time depending on the feedback the agent receives from the external world.

5.6 Conclusion

In Chapter 4, we have described the importance of the representation in the context of the generation of new objects, and argued that the value function used in deep generative modeling literature is contradictory with novelty generation. Here, we proposed KRV, a conceptual framework for clarifying the relationship between

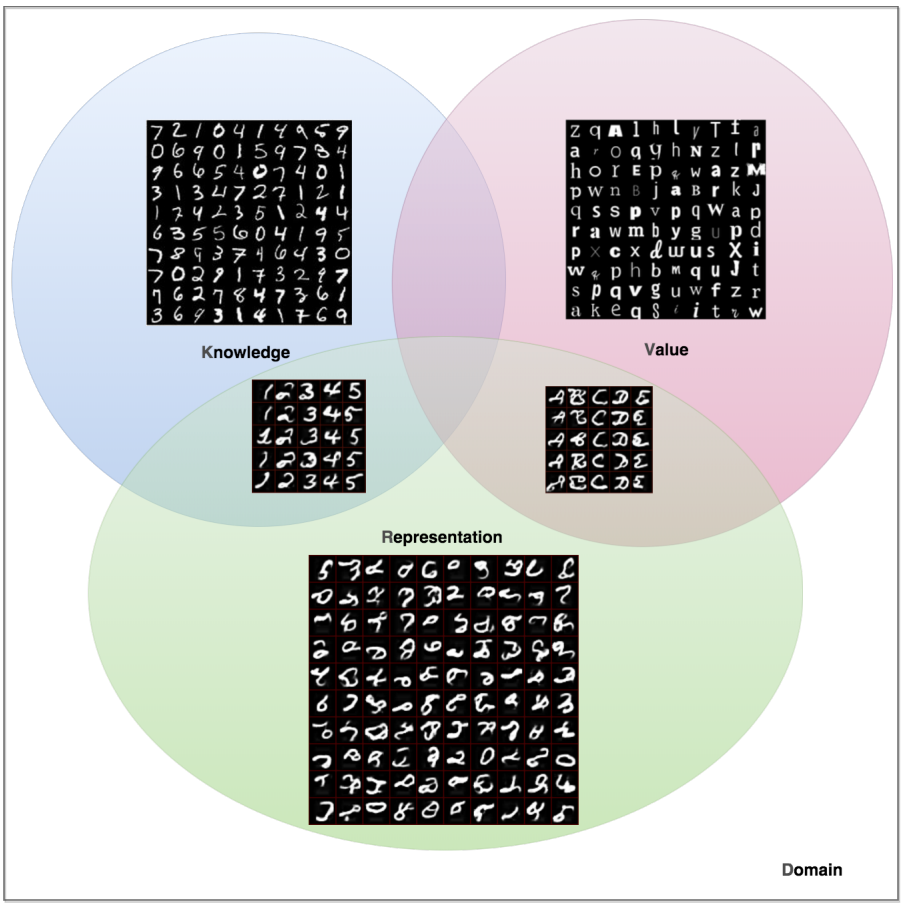


Figure 5.8: The components of the KRV framework when the external value function is given a *posteriori* to the agent. In this example, K is a set of handwritten digits. V is the set of letters. $K \cap R$ are digits generated by the agent. $V \cap R$ are letters generated by the agent. $R \setminus (V \cup K)$ are non-digit and non-letter symbols generated by the agent. All the images that belong to the R set have been generated by a sparse autoencoder, trained on MNIST.

(a) K (b) V (c) $R \cap V$

Figure 5.9: Examples of K (knowledge set), V (value set) and $R \cap V$ (generated by the agent and valuable) when an external value function is given a *posteriori* to the agent. The images shown in the third column ($R \cap V$) have been generated using a model similar than the one in Chapter 7, which is a sparse autoencoder. First, the model is trained on the MNIST dataset (handwritten digits). Then, a set of images are generated from the model. Finally, a subset of those images are selected by the probability of being a letter (each column refers to a specific letter, and the samples are ranked by probability of that letter, the top one is the highest) based on a letter classifier trained on a letter dataset from Cohen et al. (2017).

knowledge, representation and value. Although we did not give any analytical expression, we demonstrated with examples that, had we have such analytical functions, whether a priori or a posteriori, the results on the generative power of the models are dramatically different than the traditional value functions such as the likelihood. In the Chapters 7, 8 and 9, we describe some of those tasks in more detail and implement two different tasks among the ones that we proposed in this chapter and show the experimental results.

Chapter 6

The benefits of deep representations

In this chapter, we will provide arguments why deep representations can be helpful for novelty generation. The key advantage of deep nets is that they are able to learn a distributed and hierarchical representation. We argue that deep nets can not only to learn how to represent a given domain given a set of objects belonging to that domain, but also to serve a basis for generating new combinations of those learned representations, thanks to the distributed representation which allows us to describe existing objects but also new objects using the same learned features. We shall also argue that, when such a new combination is generated by a deep generative net, it is less likely to generate noise - compared, for instance, to other generative systems used in the evolutionary literature, thanks to the hierarchical representation which makes the layers mutually constrained and avoid the generation of arbitrary combinations of the features which can result in "noise". In the following we will explain more in detail those claims.

6.1 Deep nets can learn useful representations

First, as a representation learning device, a deep neural net can be seen as a way to model a domain, or a dataset. In other words, the features which are learned are useful or have a *value*, because they have the ability to *describe* examples from the domain. When trained for classification, deep nets discover the features that are *useful* to discriminate between the categories/classes. Similarly, when deep nets learn to generate data, they discover features that are *useful* to to build examples from the domain. The first characteristic of deep nets is their ability to learn distributed representations, which we define and explain in Section 6.2. The second

characteristic of deep nets is their ability to learn a hierarchy of features, which we define and explain in Section 6.3.

6.2 Distributed representations

Distributed representations (Hinton, 1984) are representations which encode each object using a set of features, and those same features are used to represent different objects. It is called *distributed* because what encodes a specific object is the *pattern of activations* of all the features simultaneously. It is contrasted with non-distributed representations or *local representations*, where the relationship between objects and features is one-to-one or one-to-few. That is, each object is defined by exactly one or few features, and those features are mutually exclusive, see Figure 6.1 for an illustration. An example of a distributed representation would be the following. Suppose we have a dataset of images of people faces, we can imagine that we could represent each image using a set of features like: gender, face shape, hair type, eyes type, eyes color, etc. Those features are not mutually exclusive and they collectively represent a face. On the other hand, a local representation would need one feature for each combination of gender, face shape, hair type, eyes type, eyes color, and would need an exponential number of features. Thus, one key advantage of distributed representations is that as the features are not mutually exclusive, they can represent an exponential number of objects, e.g., with N features we can represent 2^N if the features are binary, whereas for local representations, the number of objects we can represent are linear in terms of the number of features, where each feature can be seen as a detecting a kind of object through a *template*. In practice, the individual features do not always have a human interpretable meaning, but it is not strictly required that the features have meaning for a representation to be considered distributed. We can see in Figure 6.3 a visualization of a distributed representation of images, learned in a supervised way, which can cluster similar objects together.

Another advantage of distributed representations is that they can generalize automatically to objects never seen in the following sense. For instance, if we have a classification task which uses a distributed representation of the objects, and we are given a completely new object which have a similar (but not necessarily the same) pattern of activations than a known object then the prediction will also be similar to that known object. A notable example of this behavior occurs in *word embeddings* (Bengio et al., 2003; Mikolov et al., 2013). A word embedding (see Figure 6.2) is a way to learn a continuous and distributed representation of words in an unsupervised way. This representation can typically be used for tasks like text classification. The result of learning word embeddings is that words which share

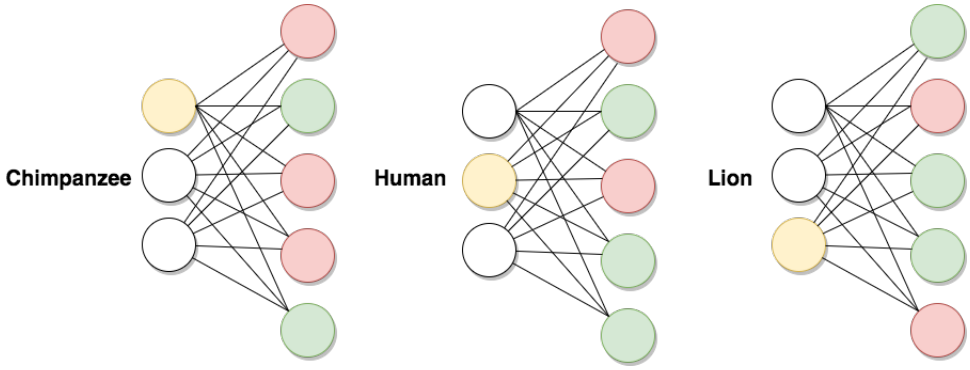


Figure 6.1: Distributed representation of 3 concepts: Chimpanzee, Human, and Lion. For each concept the left column of units (represented by circles) corresponds to a one-hot representation (only one unit is active) and the right column of units corresponds to the corresponding distributed representation. All the three concepts use the same set of features (five of them), but each concept have a different pattern of activations, which are denoted by red color when a feature is *active* and green color when it is not active. This diagram also shows that Chimpanzee and Humans are close to each other and relatively far from Lion, in the distributed representation space.

similar contexts (surrounding words) like synonyms have a similar representation (similar pattern of activations), thus even if we encounter a new text to classify, it can be related to the set of texts we previously encountered because similar words will be close to each other, e.g., the two sentences "I like eating apples" and "She loves devouring fruits" can be related to each other even if they have no words in common. We can even generalize to words we never encountered in the labeled dataset as those will have similar representation to words which have been encountered. This behavior can be contrasted with *symbolic representation* of words which does not provide a semantic similarity space, as the words are represented by one-hot[1] encoding, which implies that all words have the same distance to each other.

Importantly, in the context of generation, distributed representations are helpful because they can learn a set features that form a many to many relationship with examples from the training set, making it possible to re-use the those same features to describe new patterns of activation of the features, not seen in the training set.

[1] a high-dimensional vector with the value 1 on a specific dimension, and 0 elsewhere.

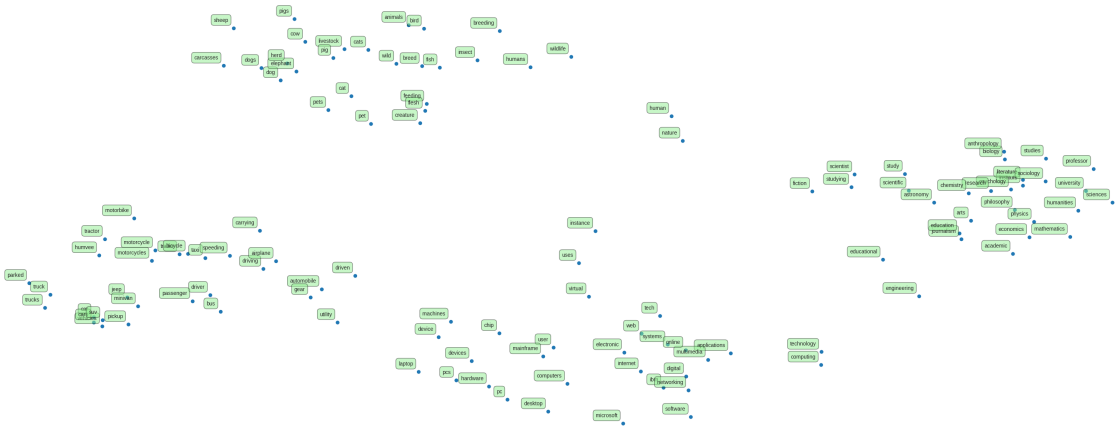


Figure 6.2: Word embeddings from Glove (Pennington et al., 2014). Each word is represented by a vector, originally in 100 dimensions, but for visualization it was projected into 2D using PCA (Wold et al., 1987). We can see four main clusters related to animals (top), vehicles (bottom left), computers (bottom middle), science (bottom right).

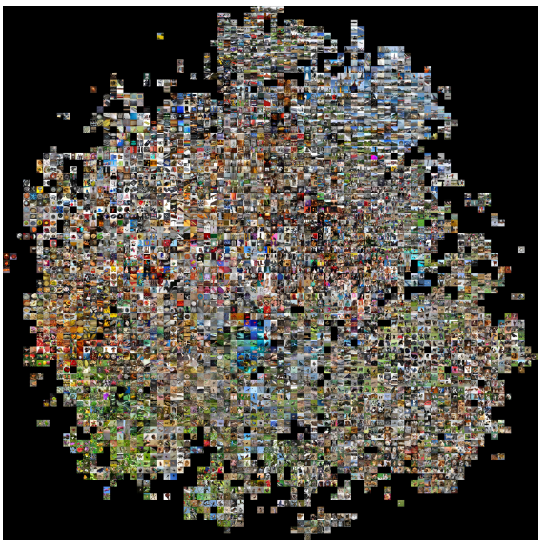


Figure 6.3: Distributed representation of images from here, initially in 4096 dimensions projected to 2D using T-SNE (van der Maaten & Hinton, 2008). Each image is blitted into its coordinates in the 2D projection.

6.3 Hierarchical compositionality

Hierarchical compositionality exists in nature. Complex livings are composed of organs, which are themselves composed of cells, which are themselves composed of molecules, which are themselves composed of atoms, and so on. Whether the fact that we represent the external word around us as a hierarchy is revealing more about the brain or the nature itself is a difficult question, but in both cases compositionality can be considered as an efficient compression mechanism, which can also give us the ability to represent and imagine meaningful new objects due to the reuse of a limited set of primitives. Indeed, one notable example which is not nature and where we can see hierarchical composition is in the structure of programming languages, where we have a set of primitives (e.g., assignments, loops, conditions), ways to combine the primitives into a function (function definition), and a way to reuse the functions inside other ones to solve new problems (function calls).

In the context of deep learning, hierarchical compositional features are features that are organized into a hierarchy of layers, from low-level layers to high-level ones, where the features in each layer are defined in terms of the features of the previous layer. Thus, what we obtain is a set of distributed representations (each layer is a representation) of the same object with growing abstraction. There are two advantages of hierarchical compositional features. The first is the ability to learn abstract or high-level features. As we described earlier, it might be helpful to represent image faces using a distributed representation with features like gender, face shape, hair type, eye type, etc. However, those features are abstract or high-level, in the sense of requiring a complex mapping from the low-level raw representation of images which consist in pixel intensities. For this reason, computing those abstract features from an image represented with a low-level representation based on pixel intensities require a complex function, which we choose to organize into layers because we assume our objects are compositional.

The second key advantage is the reusability of features. Features in each layer are defined as a *composition* of features of a lower-level layer. What makes a layer "higher-level" than another layer is exactly the fact that we are composing features of the lower-level layer together to obtain a new set of features. Feature reuse also suggests a compression benefit, because we are not relearning lower-level features each time we need to build a higher-level layer. These notions of abstraction and composition are not specific to deep learning and can be observed in multiple domains. One of the main principles of software engineering is to build reusable code (Krueger, 1992), e.g., by building functions or procedures as abstraction units which themselves can internally call other functions and procedures. As we can see in Figure 6.4, in the context of images, one can imagine a first layer detecting

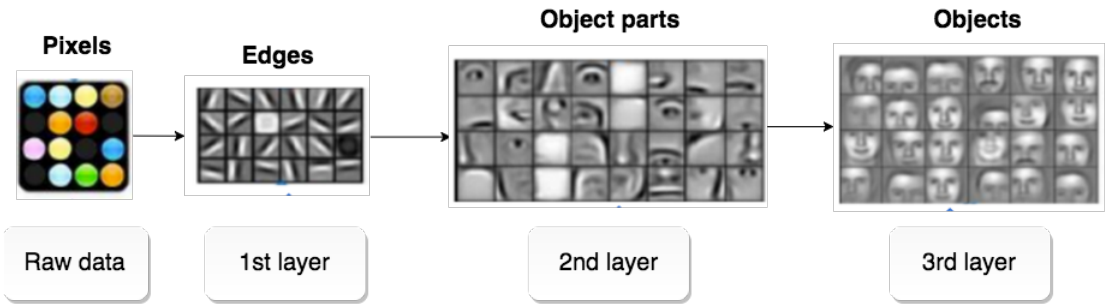
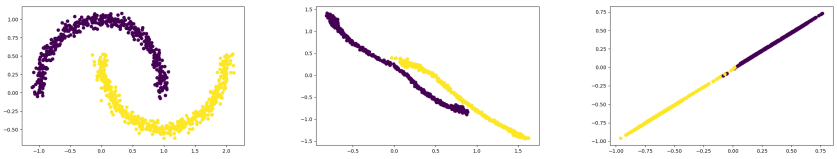


Figure 6.4: Example of feature hierarchy learned by a deep learning model on faces from Lee et al. (2009).

low-level features like edges of different orientations, then a second layer which combine edges to detect object parts like eyes, nose and mouth, and a third layer which combines object parts to detect objects.

A different argument about the importance of hierarchy for generation was given by (Bengio et al., 2013a). Several machine learning models require Markov chain Monte Carlo techniques (MCMC) to sample from the model, e.g., Deep Boltzmann machines (DBM) (Salakhutdinov & Hinton, 2009), or denoising autoencoders (Bengio et al., 2013b). An important aspect of Markov chains is the mixing speed, which can be defined as the speed (number of steps) by which it can jump across different modes of the data distribution, that is, regions of the data with high probability density. In particular, mixing can be become hard when regions of high probability density are separated by regions of low probability density. A common hypothesis made in machine learning is that the natural classes of the data distribution belong to concentrated low dimensional manifolds separated by large low density regions. As Markov chains operate in terms of local stochastic moves in the space towards more likely points, it might be very difficult to go from a high probability density region to another when they are separated by large low density regions, because it is an unlikely move. One of the questions that the paper (Bengio et al., 2013a) attempted to answer is what exactly makes samples of better quality (e.g., visually sharp in the case of images) when we use a high-level layer rather than a low-level one and what is the effect of using a high-level layer on the speed of mixing of the Markov chain.

The answer they provide is the following. First, when using a high-level layer, the data distribution in the input space is projected into a space where the manifolds containing the data are *unfolded*, that is, they become more flat (see Figure 6.5). The result of the flatness is that interpolating linearly between two likely points using the higher-level space result also in likely points, contrary to interpolating



(a) Raw representation (b) First layer (c) Second layer

Figure 6.5: A simple setup to visualize the *unfolding* effect of using layers on the manifold of the data distribution. Here the input has 2 dimensions and two classes which are non-linearly separable. A neural network with 2 hidden layers has been trained for classification. Each layer has 2 hidden units, which are visualized directly in the graphs as x and y axes. We can see that the manifold is flattened in the second layer which implies that it is possible to move along the manifold by following a line without leaving it, contrary to the raw the representation.

between two likely points in the input space which results in unlikely points (e.g., unrealistic images for image generation). The second effect of using a high-level layer is that the relative volume which is occupied by likely points is expanded (see Figure 6.6), which means that in a higher-level feature space the probability of finding unlikely points is reduced, even when moving on random directions (see Figure 6.6). It also makes it easier to establish bridges between modes (thus simplifying mixing between them), which results in likely points when interpolating linearly between one point from a mode and another point from another mode (e.g., interpolating an image from one category to an image from another category).

In summary, in a hierarchical representation, layers are mutually constrained. Because they are composed together, the features of a layer constrain the value of the features of the next layer. Equivalently, the values of the features of a layer are constrained by the value of the features of the previous layer. As a result, the pattern of activations of the features of any layer in the hierarchy cannot be arbitrary. Those *learned* constraints are helpful, because in a generation context, they force the output of the generative model to be constrained as well[2], making it unlikely to generate "noise". As an example, suppose we learned to encode handwritten digits using a set of strokes and their positions in the image. Suppose also we learned a decoder that transforms the strokes and their positions into pixel intensities. Here, the images generated through the decoder cannot be arbitrary, because the decoder is constrained to compose the stroke features in such a way that only strokes can appear in the output image.

[2]The output of the model is just one layer among the others.

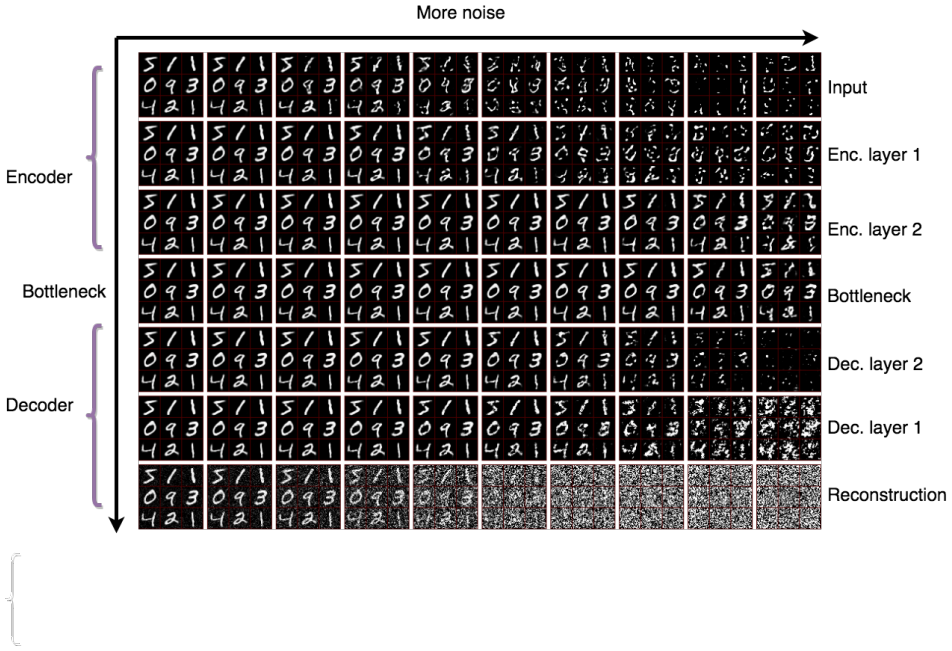


Figure 6.6: Illustration of the relative expansion of the volume taken by likely points in deeper layers discussed in Section 6.3. A deep convolutional autoencoder has been trained on MNIST. Each row corresponds to a particular layer, whereas each column corresponds to a particular amount of white noise added to the layer activations. To obtain the 9 images corresponding to each couple (layer, amount of noise), we use 9 images from MNIST then obtain the activations of the layer, then add noise to the activations, then reconstruct. We can see that the closer the layer is to the bottleneck layer, the harder it is to destroy the images structure even by adding noise in random directions.

6.4 Comparison to evolutionary systems

We described in Section 6.2 and Section 6.3 two important characteristics of deep learning models, and why they can be helpful for novelty generation. Researchers in the evolutionary literature (Togelius et al., 2011) argue that representation is a central question in stochastic optimization and meta-heuristics, as changing the nature of the search space can completely change the difficulty of the problem. Evolutionary systems are considered to be a kind of *generate-and-test* (Togelius et al., 2011) systems, where a population of objects are evolved to generate new objects through mutation (random perturbation) and combination (crossover) on the chosen representation space of objects, then the best individuals are selected through a fitness function which reflects the preference system of the *designer*. In evolutionary systems, we usually distinguish two representation spaces, the genotype and the phenotype. The *genotype* is the representation where the *optimization* (e.g., through mutation and crossover) is performed, whereas the *phenotype* is where the *evaluation* is performed, and the two need not to be the same. For instance, the genotypes of a simulated robot could represent the parameters of the controller and the shape of the robot, whereas the phenotype is the actual behavior of the robot in the simulation, which is what is actually used by the fitness function to evaluate the robot. Depending on how much the mapping from the genotype to the phenotype is complex, we might talk about direct or indirect encodings. The most direct encoding would be to use the raw representation of the objects (e.g., pixels for images, time-frequency for sounds) for both the genotype and the phenotype. An indirect encoding would be to use a generator which takes a small number of parameters (genotype) as input and outputs the objects in their raw representation (phenotype), and the goal would be to only optimize a small number of parameters.

The main issue with direct encodings is that the space is usually extremely large (e.g., pixel space, or time-frequency space) and most objects in that space are non-meaningful (e.g., noise). In that case, evolutionary systems lack a model of knowledge. This is why indirect encodings with generators like the Compositional pattern-producing Network (CPPN) (Stanley, 2007) are used, and those can be considered as a model of knowledge. With CPPN (Stanley, 2007) specifically, what is evolved are the weights of a neural network instead of the pixels themselves (indirect encoding), and the neural network is used to generate a single image. However, just like feature engineering, indirect encodings are hand-crafted for each specific task and they are not *data-driven*. For that reason, the result of the optimization will heavily depend on how much powerful (or limited) is the hand-crafted system. On the other hand, in deep learning systems it is possible to learn automatically hierarchical distribution representations (which correspond to an indirect encoding) in a data-driven way, giving it an adaptive behavior.

6.5 Conclusion

In this chapter, we have described the advantages deep nets offer for novelty generation. We argued that deep nets can be beneficial thanks to the combined effects of distributed and hierarchical representations they learn. With a distributed representation, deep nets can learn a set of features that can describe well a given domain or a dataset, and can use those same features to describe new examples. With a hierarchical representation, the features of the layers are mutually constrained making the output of the generative model not arbitrary, limiting the probability of generating noise. The two effects combined give one of the most desirable characteristics of deep nets for generating novelty: the representations learned are value-preserving in the sense that they are useful to describe not only data that the net has seen during training but also unseen data that could have been generated by the same features.

In this chapter, we have also compared deep nets to models optimized with evolutionary algorithms, and argued that they either lack a model of knowledge or use a limited hand-coded model of knowledge, whereas deep nets are flexible and data-driven. The main takeaway message from this comparison is tied to the preservation of value: while small changes to the raw representations evolutionary algorithms use bring about noise almost immediately, generating from the dense representations a deep net learns allows sustaining perturbations more easily without yielding noise. In the coming chapters, we will describe a set of experiments and results of using deep nets for novelty generation.

Part III

Experiments on novelty generation

Chapter 7

Digits that are not: Generating new types through deep neural nets

In this chapter, we argue that an important barrier for progress in computational creativity research is the inability of these systems to develop their own notion of value for novelty. We propose a notion of knowledge-driven value function that circumvent the need for an externally imposed value function, allowing the system to explore based on what it has learned from a set of referential objects, through an internal value function. The concept is illustrated by a specific knowledge model provided by a deep generative autoencoder. Using the described system, we train a knowledge model on a set of digit images and we use the same model to build coherent sets of new digits that do not belong to known digit types. The content of this chapter is from the paper Kazakçı, Cherti, and Kégl (2016).

7.1 Introduction

It is a widely accepted view in creativity research that creativity is a process by which novel *and* valuable combinations of ideas are produced (Runco & Jaeger, 2012). This view bears a tension, the essence of which can be expressed by the following question: how to determine the value of novelty? If a new object is substantially different of the previous objects in its category, it might be hard to determine its value. On the contrary, if the value of an object can be readily determined, it might be the case that the object is not genuinely new. Indeed, there exist experimental results positing that novelty is a better predictor of creativity than the value (Diedrich et al., 2015) and that the brain processes novelty in a particular

Figure 7.1: *Digits that are not*. Symbols generated using a deep neural net trained on a sample of hand-written digits from 0 to 9.

way (Beaucousin et al., 2011), suggesting that the relationship is far from trivial.

In art, the difficulty in determining the value of an object is omnipresent. An emblematic example is *Le Grand Verre* by Marcel Duchamp. The artist worked on this singular project from 1915 to 1923 and produced a groundbreaking yet enigmatic piece of art, which the critiques still continue to interpret in various ways. In 1934, Duchamp built *La boîte verte*, a green box containing preparatory material (notes, drawings, photographs) he produced for *Le Grand Verre*. Considered as a piece of art in its own right, the box was intended to assist and to explain *Le Grand Verre*, as would an exhibition catalog (Breton, 1932).

In product design, there exist less enigmatic but still emblematic cases, where the value of an innovation could not be easily determined. For instance, the first smartphone received significant criticism regarding its usability (e.g., no stylus was provided), and it was deemed to be less *evolved* than its counterparts. Beyond such problems related to the reception of novelty, the sheer difficulty in discovering new value has led companies to seek alternative approaches, such as input from lead users (Von Hippel, 1986).

The difficulty in determining the value of novelty has particular implications from a computational perspective. How would a creative agent drive his search process towards novelty if its evaluation function has been predetermined? In practical implementations, we can find various manifestations of such fixed evaluation functions such as fitness functions or quantitative aesthetics criteria. These implementations fixate the kind of value the system can seek, once and for all in the beginning of the process. The creative outcome, if any, comes from an output whose perception was unexpected or unpredictable.

Theoretically, it may be argued that this can be solved by allowing the creative agent to change its own evaluation rules (Wiggins, 2006; Jennings, 2010). This implies that the system would be able to develop a preference for unknown and novel types of objects (Kazakçı, 2014). In practice, this is implemented by interactive systems that use external feedback (e.g., the preferences of an expert) to guide the

search process. Such systems explore user preferences about novelty rather than building their own value system. This is a shortcoming from the point of view of creativity (Kazakçı, 2014).

An alternative approach might be to force the system to systematically explore *unknown* objects (Hatchuel & Weil, 2009). This requires the system to function in a differential mode, where there is a need to define a reference of *known* objects. In other words, new kinds of values might be searched by *going-out-of-the-box* mechanisms which require the system to develop knowledge about a referential set of objects. In the absence of knowledge about such a set, creativity is reduced either to a combinatorial search or to a rule-based generative inference, both of which explore boundaries confined by the creator of the system and not the system itself. When such knowledge exists, the system can explore new types of objects by tapping into the *blind spots* of the knowledge model (Kazakci, Akin et al., 2010).

In this paper, we use a deep generative neural network to demonstrate knowledge-driven creativity. Deep nets are powerful tools that have been praised for their capacity of producing useful and hierarchically organized representations from data. While the utility of such representations have been extensively demonstrated in the context of recognition (i.e., classification) far less work exists on exploring the generative capacity of such tools.

In addition, the goal of the little work on generative deep nets is to generate objects of *known types*, and the quality of the generator is judged by the visual or quantified similarity with existing objects (e.g., an approximate likelihood) (Theis et al., 2015). In contrast, we use deep nets to explore their generative capacity beyond known types by generating unseen combinations of extracted features, the results of which are symbols that are mostly unrecognizable but seemingly respecting some implicit semantic rules of compositionality (Figure 7.1). What we mean by *features* is a key concept of the paper: they are not decided by the (human) designer, rather learned by an *autoassociative* coding-decoding process.

The novelty of our approach is two-fold. With respect to computational creativity models, our model aims at explicitly generating new types. We provide an experimental framework for studying how a machine can develop its own value system for new types of objects. With respect to statistical sample-based generative models, rather than a technical contribution, we are introducing a new *objective*: generate objects that are, in a deep sense, similar to objects in of the domain, but which use learned *features* of these objects to generate new objects which do not have the same *type*. In our case, we attempt to generate images that *could be* digits (e.g., in another imaginary culture), but which are *not*.

The first section, *Generative models for computational creativity*, describes our positioning with respect to some of the fundamental notions in creativity re-

search in previous works. The section *Learning to generate* presents details about data-driven generative models and deep neural nets relevant to our implementation. The section *Generating from the learned model* describes our approach for exploring novelty through generation of new types, presents examples and comments. The section *Discussion and perspectives* discusses links with related research and points to further research avenues. Finally, section *Summary* concludes.

7.2 Generative models for computational creativity

7.2.1 The purpose of a generative model

In the computational creativity literature, exploration of novelty has often been considered in connection with art (Boden & Edmonds, 2009; McCormack et al., 2014). Despite various debates and nuances on terminology, such work has generally been categorized under the term *generative art* (or generative models). As defined by (Boden & Edmonds, 2009), a generative model is essentially a rule-based system, albeit one whose output is not known in advance, for instance, due to non-determinism or to many degrees of freedom in the parameters of the systems (see also (Galanter, 2012a)). A large variety of such systems has been built, starting as early as the 90s (Todd & Latham, 1991; Sims, 1991), based on even earlier foundations (Nees, 1969; Edmonds, 1969). The definition, the complexity and the capabilities offered by such models evolved consistently. To date, several such models, including L-systems, cellular automata, or artificial life simulations, have been used in various contexts for the generation of new objects (i.e., drawings, sounds, or 3D printings) by machine. Such systems achieve an output perceived as creative by their users by opportunistically exploiting existing formal approaches that have been invented in other disciplines and for other purposes. Within this spirit, computational creativity research has produced a myriad of successful applications on highly complex objects, involving visual and acoustic information content.

In contrast, this work considers much simpler objects since we are interested, above all, in the clarification of notions such as novelty, value, or type, and in linking such notions with the solid foundations of statistics and machine learning. These notions underlie foundational debates on creativity research. Thus, rather than producing objects that might be considered as artistic by a given audience, our purpose is to better define and explicate a minimalist set of notions and principles that would hopefully lead to a better understanding of creativity and enable further experimental studies.

7.2.2 The knowledge of a generative system

The definition of a generative model as a rule-based system (Boden & Edmonds, 2009) induces a particular relationship with knowledge. It is fair to state that such formalized rules are archetypes of consolidated knowledge. If such rules are hard-coded into the creative agent by the system designer, the system becomes an inference engine rather than a creativity engine. By their very nature, rules embed knowledge about a domain and its associated value system that comes from the system designer instead of being discovered by the system itself.

Allowing the system to learn its own rule system by examining a set of objects in a given domain resolves part of this problem: the value system becomes dependent on the learning algorithm (instead of the system designer). In our system, we use a learning mechanism where the creative agent is forced to *learn to disassemble and reconstruct* the examples it has seen. This ensures that the utility of the features and the transformations embedded within the rules learned by the system are directly linked to a capacity to construct objects. As we shall see, the particular deep neural net architecture we are using is not only able to *re*construct *known* objects: it can also *build new* and *valuable* objects using their hierarchically organized set of induced transformations.

7.2.3 Knowledge-driven exploration of value

Today, more often than not, generative models of computational creativity involve some form of a biological metaphor, the quintessence of which is evolutionary computation (McCormack, 2013). Contrary to human artists who are capable of exploring both novelty and the value of novelty, such computational models often consider the generation of novelty for a value function that is independent of the search process. Either they operate based on a fixed set of evaluation criteria or they defer evaluation to outside feedback. For the former case, a typical example would be a traditional fitness function. For the later case, a typical example would be an interactive genetic algorithm (Takagi, 2001) where the information about value is provided by an oracle (e.g., a human expert). In both cases, the system becomes a construction machine where the generation of value is handled by some external mechanism and not by the system itself. This can be considered as a fundamental barrier for computational creativity research (Kazakçı, 2014) that we shall call *fitness function barrier*.

(Parikka, 2008) summarizes the stagnation that this approach causes for the study of art through computers: "*. . . if one looks at several of the art pieces made with genetic algorithms, one gets quickly a feeling of not 'nature at work' but a Designer that after a while starts to repeat himself. There seems to be a teleology*

anyhow incorporated into the supposed forces of nature expressed in genetic algorithms practice 'a vague feeling of disappointment surrounds evolutionary art'".

The teleology in question is a direct consequence of fitness function barrier and the hard-coded rules. In our system, we avoid both issues by using a simple mechanism that enables the system to explore novel objects with novel values. Given a set of *referential objects* $\mathcal{D} = \{x_1, .., x_n\}$ whose *types* $\mathcal{T} = \{t_1, ..., t_k\}$ are *known* (or can be determined by a statistical procedure such as clustering), the system is built in such a way that it generates objects $\mathcal{D}' = \{x'_1, \ldots, x'_m\}$ with types $\mathcal{T}' = \{t'_1, \ldots, t'_\ell\}$ such that $\mathcal{D}' \not\subset \mathcal{D}$ and $\mathcal{T}' \not\subset \mathcal{T}$. In other words, the system builds a set of new objects, some of which have new types. While the current system does not develop a preference function over the novelty it generates, the current setup provides the necessary elements to develop and experiment with what might be a value function for the unknown types. At any rate, the generation of unknown types of objects is an essential first step for a creative system to develop its own evaluation function for novelty and to become a designer itself.

7.3 Learning to generate

7.3.1 Data-driven generative models

In contrast to computational creativity research that aims to generate new object descriptions, disciplines such as statistics and machine learning strive to build solid foundations and formal methods for modeling a given set of object descriptions (i.e., *data*). These disciplines do not consider the generation of data as a scientific question: the data generating process is considered fixed (given) but unknown. Nevertheless, these fields have developed powerful theoretical and practical formal tools that are useful to scientifically and systematically study what it means to generate novelty.

In fact, generative models have a long and rich history in these fields. The goal of generative models in statistics and machine learning is to sample from a fixed but unknown *probability distribution* $p(x)$. It is usually assumed that the algorithm is given a *sample* $\mathcal{D} = \{x_1, \ldots, x_n\}$, generated independently (by nature or by a simulator) from $p(x)$. There may be two goals. In classical *density estimation* the goal is to estimate p in order to evaluate it later on any new object x. Typical uses of the learned density are *classification* (where we learn the densities $\widehat{p}_1$ and $\widehat{p}_2$ from samples $\mathcal{D}_1$ and $\mathcal{D}_2$ of two *types* of objects, then compare $\widehat{p}_1(x)$ and $\widehat{p}_2(x)$ to decide the type of x), or *novelty* (or *outlier*) *detection* (where the goal is to detect objects from a stream which do not look like objects in $\mathcal{D}$ by thresholding $\widehat{p}(x)$).

The second goal of statistical generative models is to *sample* objects from the generative distribution p. If p is known, this is just random number generation.

If p is unknown, one can go through a first density estimation step to estimate $\widehat{p}$, then sample from $\widehat{p}$. The problem is that when x is high-dimensional (e.g., text, images, music, video), density estimation is a hard problem (much harder than, e.g., classification). A recent line of research (Hinton et al., 2006; Salakhutdinov & Hinton, 2009) attempts to generate from p without estimating it, going directly from $\mathcal{D}$ to novel examples. In this setup, a formal generative model g is a function that takes, as input, a random seed r, and generates an object $x = g(r)$. The learning (a.k.a, training or building) process is a (computational) function $\mathcal{A}$ that takes, as input, a data set $\mathcal{D}$, and outputs the generative model $g = \mathcal{A}(\mathcal{D})$.

The fundamental problem of this latter approach is very similar to the main question we raised about computational creativity: what is the value function? When the goal is density estimation, the value of $\widehat{p}$ is formally $\sum_{x \in \mathcal{D}'} \log \widehat{p}(x)$, the so-called *log-likelihood*, where $\mathcal{D}'$ is a second data set, independent from $\mathcal{D}$ which we used to build (or, in machine learning terminology, to *train*) $\widehat{p}$. When p is unknown, evaluating the quality of a generated object $x = g(r)$ or the quality of a sample $\widehat{\mathcal{D}} = \{g(r_1), \ldots, g(r_n)\}$ is an unsolved research question in machine learning as well.

There are a few attempts to formalize a quantitative goal (Goodfellow et al., 2014), but most of the time the sample $\widehat{\mathcal{D}}$ is evaluated visually (when x is an image) or by listening to the generated piece of music. And this is tricky: it is trivial to generate exact objects from the training set $\mathcal{D}$ (by random sampling), so the goal is to generate samples that are *not* in $\mathcal{D}$, but which *look like* coming from the *type* of objects in $\mathcal{D}$. By contrast, our goal is to generate images that look like *digits* but which do not come from digit *types* present in $\mathcal{D}$.

7.3.2 Deep neural networks

In the machine learning literature, the introduction of deep neural networks (DNNs) is considered a major breakthrough (LeCun et al., 2015). The fundamental idea of a DNN is to use of several hidden layers. Subsequent layers process the output of previous layers to sequentially transform the initial representation of objects. The goal is to build a specific representation useful for some given task (i.e., classification). Multi-layered learning has dramatically improved the state of the art in many high-impact application domains, such as speech recognition, visual object recognition, and natural language processing.

Another useful attribute of deep neural nets is that they can learn a *hierarchy* of representations, associated to layers of the net. Indeed, a neural net with L layers can be formalized as a sequence of coders $(c^1, \ldots, c^L)$. The representation in the first layer is $y^1 = c^1(x)$, and for subsequent layers $1 < \ell \leq L$ it is $y^\ell = c^\ell(y^{\ell-1})$. The role of the output layer is then to map the top representation y^L onto

a final target $\widehat{y} = d(y^L)$, for example, in the case of classification, onto a finite set of object types. In what follows, we will denote the function that the full net implements by f. With this notation, $\widehat{y} = d(y^L) = d\big(c^L(y^{L-1})\big) = \ldots = f(x)$.

The formal training setup is the following. We are given a training set $\mathcal{D} = \{x_1, \ldots, x_n\}$, a set of learning targets (e.g., object types) $\{y_1, \ldots, y_n\}$, and a score function $s(y, \widehat{y})$ representing the error (negative value) of the prediction $\widehat{y}$ with respect to the real target y. The setup is called *supervised* because both the targets of the network y_i and the value of its output s is given by the designer. We train the network f_w, where w is the vector of all the parameters of the net, by classical stochastic gradient descent (modulo technical details): we cycle through the training set, reconstruct $\widehat{y_i} = f_w(x_i)$, compute the gradient $\delta_i = \partial s(y_i, \widehat{y_i})/\partial w$, and move the weights w by a small step in the direction of $-\delta_i$.

7.3.3 Autoassociative neural nets (autoencoders)

Formally, an autoencoder is a supervised neural network whose goal is to predict the input x itself. Such neural networks are composed of an encoder part and a decoder part. In a sense, an autoencoder learns to disassemble then to reassemble the object x. Our approach is based on a particular the technique described in (Bengio et al., 2013b). We first learn about the input space by training an *autoassociative* neural net (a.k.a. *autoencoder*) f using objects $\mathcal{D} = \{x_1, \ldots, x_n\}$, then apply a technique that designs a generative function (simulator) g based on the trained net f.

Autoencoders are convenient because they are designed to learn a representation $y = c(x)$ of the object x and a decoder $x' = d(y)$ such that x is close to x' in some formal sense, and y is concise or simple. In the classical *information theoretical* paradigm, both criteria can be formalized: we want the code length of y (the number of bits needed to store y) to be small while keeping the distortion (e.g., the Euclidean distance) between x and x' also small. In (neural) *representation learning*, the goals are somewhat softer. The distortion measure is usually the same as in information theory, but simplicity of y is often formalized implicitly by using various *regularization* operators. The double goal of these operators is to prevent the algorithm to learn the identity function for the coder c, and to learn a y that uses elements ("code snippets") that agree with our intuition of what object components are.

The decoder d takes the top representation y^L and reconstructs $x' = d(y^L)$. The goal is to minimize a score $s(x, x')$, also called distortion, that measures how close the input image x is to the reconstructed image x'. Throughout this paper, we will use the Euclidean squared distance in the pixel space $s(x, x') = \|x - x'\|_2^2$.

We are using a particular variant of autoencoders, called sparse convolutional

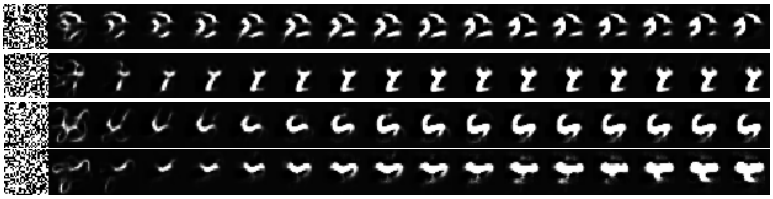


Figure 7.2: Four examples illustrating the iterative generative process. At each iteration, the net pushes the input image closer to what it can "understand" (reconstruct easily), converging to a fixed point (an image that can be reconstructed without an error).

autoencoders (Makhzani & Frey, 2015) with $L = 3$ coding layers and a single decoding layer. Convolutional layers are neural net building blocks designed specifically for images: they are essentially small (e.g., 5×5) filters that are repeated on the full image (in other words, they share the same set of weights, representing the filter). The sparse regularizer penalizes dense activations, which results in a sparse representation: at any given layer, for any given image x, only a small number of units ("object parts", elements of y^ℓ) are turned on. This results in an interesting structure: lower layer representations are composed of small edgelets (detected by Gabor-filter like coders), followed by small object parts "assembled" from the low-level features. The convolutional filters themselves are object parts that were extracted from the objects of the training set. The sparsity penalty and the relatively small number of filters force the net to extract features that are general across the population of training objects.

7.4 Generating from the learned model

In this section we present and comment some experimental results. First, we provide some illustrations providing an insight regarding the usefulness of the representations extracted by a deep net for searching for novelty. Then, we present the method we use to generate novel image objects, based on the formal approach described in the section *Learning to generate*.

7.4.1 Searching for new types: with and without knowledge

We argued in previous sections that combinatorial search over the objects has disadvantages over a search process driven by a knowledge over the same set of objects obtained by the system itself. When the learning is implemented through a deep neural net, this knowledge is encoded in the form of multiple levels of representations and transformations from layer to layer. To demonstrate the effect of

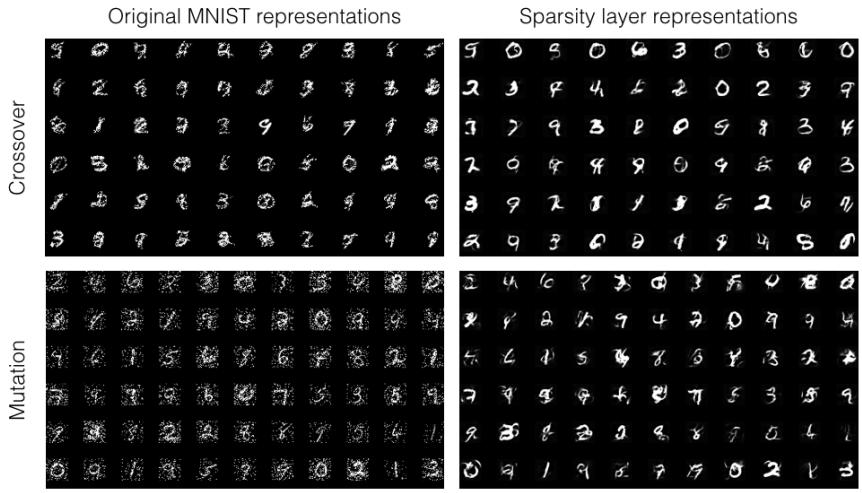


Figure 7.3: The effect of perturbations applied to object representations. On the left, the effect of crossover and mutation on the original representations of MNIST. On the right, the same operators applied to the representations learned by the deep generative net. Visually, this latter category seem less affected by perturbations, and thus is likely to provide a better search space for novelty.

knowledge over these search procedures, instead of searching in the original object space of x, we have applied simple perturbation operations on the representation space y.

Figure 7.3 illustrates the results of these perturbations. In the original representation space, crossover and mutation operators create noisy artifacts, and the population quickly becomes unrecognizable, which, unless the sought effect is precisely the noise, is not likely to produce novel objects (let alone types) unless a fitness function that drives the search is given (which is what we are trying to avoid). In comparison, the same operators applied to the code y produced by the deep nets produce less noisy and seemingly more coherent forms. In fact, some novel symbols that go beyond the known digits seem to have already emerged and can be consolidated by further iteration through the model. Overall, combinatorial search in the representation space provided by the deep net seems more likely to generate meaningful combinations in the absence of a given evaluation function, thus, making it more suitable for knowledge-driven creativity.

7.4.2 Method for generating new objects from a learned model

To generate new objects in a knowledge-driven fashion, we first train a generative autoencoder to extract features that are useful for constructing such objects. To

train the autoencoder f, we use the MNIST (Lecun & Cortes) data set (Figure 7.4) containing gray-scale hand-written digits. It contains 70 000 images of size 28×28. Once the model learned to construct objects it has seen, it has also learned useful transformations that can be queried to generate new objects.

Figure 7.4: A subsample of MNIST, the data set we use to train the autoencoder f.

Autoassociative networks exist since the 80s (Rumelhart et al., 1986; Baldi & Hornik, 1989; Kramer, 1991), nevertheless, it was discovered only recently that they can be used to generate new objects (Bengio et al., 2013b; Kamyshanska & Memisevic, 2013). The procedure is the following. We start from a random image $x_0 = r$, and reconstruct it $x_1 = f(x)$ using the trained network f. Then we plug the reconstructed image back to the net and repeat $x_k = f(x_{k-1})$ until convergence. Figure 7.2 illustrates the process. At each step, the net is forced to generate an image which is easier to reconstruct than its input. The random seed r initializes the process. From the first iteration on, we can see familiar object parts and compositions rules, but the actual object is new. The net converges to a fixed point (an image that can be reconstructed without an error).

It can be observed that, although this kind of generative procedure generates new objects, the first generation of images obtained by random input (second column of Figure 7.2) look noisy. This can be interpreted as the model has created a novelty, but has not excelled yet at constructing it adequately. However, feeding this representation back to the model and generating a new *version* improves the quality. Repeating this step multiple times enables the model to converge effectively towards fixed points of the model, that are more precise (i.e., visually). Their novelty, in terms of typicality, can be checked using clustering methods and visualized as in Figure 7.5.

7.4.3 Generating new types

When the generative approach is repeated starting from multiple random images $\{r_1, \ldots, r_n\}$, the network generates different objects $\{x_1, \ldots, x_n\}$. When projecting these objects (with the original MNIST images) into a two-dimensional space using stochastic neighbor embedding (van der Maaten & Hinton, 2008), the space

is not filled uniformly: it has dense clusters, meaning that structurally similar objects tend to regroup; see Figure 7.5. We recover these clusters quantitatively using k-means clustering in the feature space $\{y_1, \ldots, y_n\}$. Figure 7.6 contains excerpts from these clusters. They are composed of similar symbols that form a coherent set of objects, which can be perceived as new *types*.

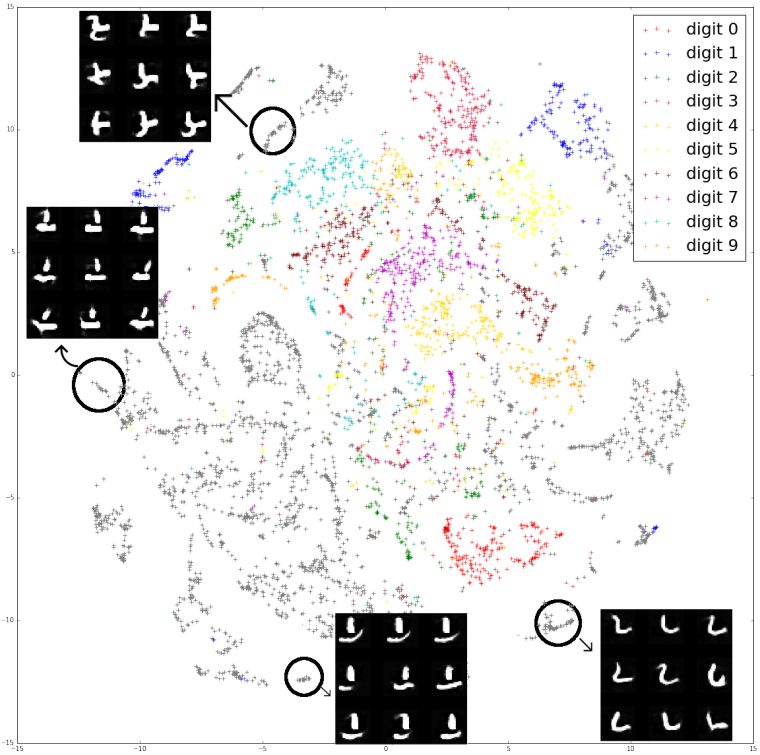


Figure 7.5: A distance-preserving projection of digits to a two-dimensional space. Colored clusters are original MNIST types (digit classes from 0 to 9). The gray dots are newly generated objects. Objects from four of the clusters are displayed.

7.5 Discussion and perspectives

It is possible to compare our work with several other published results. To start with, the generation of novelty through the use of neural nets is an old idea (Todd, 1992, 1989; Thaler, 1998). There are two main differences between our approach and theirs. First, our emphasis is on studying how an artificial agent can generate novelty that does not fit into learned categories, rather than creating objects with

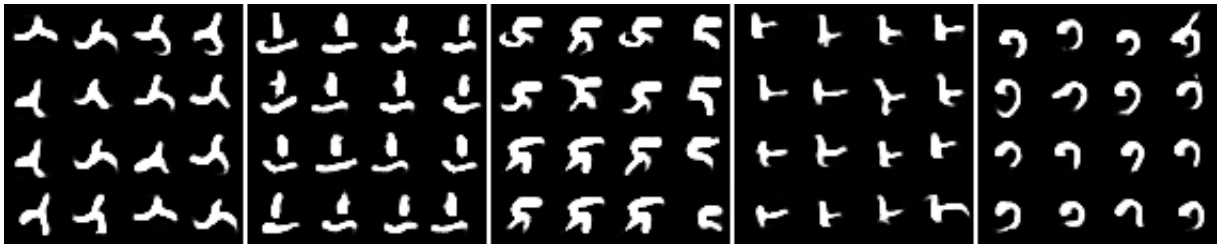


Figure 7.6: A sample of new types discovered by the model

artistic value. This experimental setup is intended to provide means for studying how a creative agent can build an evaluation function for new types of objects. Second, we explicitly aim at establishing a bidirectional link between generative models for computational creativity and generative models within statistics and machine learning. Beyond the use of techniques and tools developed in these disciplines, we wish to raise research questions about creative reasoning that would also be interesting in statistics and machine learning.

In fact, some recent work has already started exploring the creative potential of deep neural networks. For instance, (Mordvintsev et al., 2015) uses a deep net to project the input that would correspond to a maximal activation of a layer back onto an image in an iterative fashion. The images are perceived as dreamy objects that are both visually confusing and appealing. Another work (Gatys et al., 2015a) uses correlations of activations in multiple layers of a deep net to extract style information from one picture and to transpose it to another. Finally, (Nguyen et al., 2015b) uses a trained net as a fitness function for an evolutionary approach (see also (Machado et al., 2008) for a similar application with shallow nets). These successful approaches demonstrate the potential of deep nets as an instrument for creativity research and for generating effects that can be deemed as surprising, even creative. The present approach and the points the paper puts forward are significantly different. Compared to the architectures used in these studies, ours is the only one that uses a generative deep autoassociative net. The reason for this choice is twofold. First, we aim at using and understanding the generative capacity of deep nets. Second, we are interested in the deconstruction and reconstruction our architecture provides since our aim is to build objects through the net (not to create an effect that modifies existing objects). Once again, thinking about and experimenting with these foundational aspects of generative deep nets provide a medium through which notions of creativity research can be clarified through statistical notions. This is not among the declared objectives of previous works.

The novelty-seeking behavior of our system can also be compared to the recent novelty-driven search approaches in the evolutionary computing literature (Mouret

& Doncieux, 2012; Lehman & Stanley, 2011b). These approaches, like ours, seek to avoid objective functions and push the system to systematically generate novelty in terms of system behavior (e.g., novelty in the output). Our system is akin to such methods in spirit with one main difference: we believe that knowledge plays a fundamental role in creative endeavor and the decision of the system regarding the search for novelty should come from its own knowledge model. Note that this does not exclude a more general system where several systems such as ours can compete to differentiate themselves from the observed behavior of others, effectively creating a community of designers.

Our system provides a step towards an experimental study of how an artificial agent can drive its search based on knowledge. Furthermore, it can effectively create new types of objects preserving abstract and semantic properties of a domain. However, we have not fully addressed the question of how such an agent can build its own value function about novelty. Nevertheless, the system enables numerous ways to experiment with various possibilities. An obvious next step would be to hook our system to an external environment, where the system can receive feedback about value (Clune & Lipson, 2011; Secretan et al., 2008). To avoid the fitness function barrier, this should be done in such a way that the system can build its own value system rather than only learning the ones in its environment.

7.6 Conclusion

We provided an experimental setup based on a set of principles that we have described. The pinnacle of these principles is that artificial creativity can be driven by knowledge that a machine extracts itself from a set of objects defining a domain. Given such knowledge, a creative agent can explore new *types* of objects and build its own value function about novelty. This principle is in contrast with existing systems where the system designer or audience imposes a value function to the system, for example, by some fitness function.

We argued that when an artificial creative agent extracts its own domain knowledge in the form of features that are useful to reconstruct the objects of the domain, it becomes able to explore novelties beyond the scope of what it has seen by exploring systematically unknown types. We have demonstrated the idea by using a deep generative network trained on a set of digits. We proposed a compositional sampling approach that yielded a number of new types of digits.

While our setup provides a basis for further exploring how an agent can develop its own value function, it is also a bridge with the powerful theories and techniques developed within the statistics and machine learning communities. A colossal amount of work has already been published on deep neural networks with

significant breakthroughs in many domains. Deep learning will be all the more valuable if it offers an evolution of the machine learning paradigm towards machine creativity.

Chapter 8

Out-of-class novelty generation: an experimental foundation

In this chapter, we propose an external value function that judges the capability of models to generate novelty a posteriori. We assess several metrics designed for evaluating the quality of generative models on this new task. We also propose a new experimental setup. Inspired by the usual held-out validation, we hold out entire classes for evaluating the generative potential of models. The goal of the novelty generator is then to use training classes to build a model that can generate objects from future (hold-out) classes, unknown at training time - and thus, are novel with respect to the knowledge the model incorporates. Through extensive experiments on various types of generative models, we are able to find architectures and hyperparameter combinations which lead to out-of-distribution novelty. The content of this chapter is from the paper Cherti, Kégl, and Kazakçı (2016) and Cherti, Kégl, and Kazakçı (2016).

8.1 Introduction

Recent advances in machine learning have renewed interest in artificial creativity. Studies such as deep dream (Mordvintsev et al., 2015) and style transfer (Gatys et al., 2015b) have aroused both general public interest and have given strong impetus to use deep learning models in computational creativity research (ICC, 2016). Although creativity has been a topic of interest on and off throughout the years in machine learning (Schmidhuber, 2009), it has been slowly becoming a legitimate sub-domain with the appearance of dedicated research groups such as Google's Magenta and research work on the topic (Nguyen et al., 2015b; Lake et al., 2015).

There is a large body of work studying creativity by computational methods. A

large variety of techniques, from rule-based systems to evolutionary computation has been used for a myriad of research questions. Compared to these methods, machine learning methods provide an important advantage: they enable the study of creativity in relation with knowledge (i.e., knowledge-driven creativity; (Kazakçı et al., 2016)). Nevertheless, to better highlight the points of interest in computational creativity research for the machine learning community and to allow machine learning researchers to provide systematic and rigorous answers to computational creativity problems, it is important to precisely answer three questions:

1. What is meant by the generation of novelty?
2. How can novelty be generated?
3. How can a model generating novelty be evaluated?

Within the scope of machine learning, it would be tempting to seek answers to these questions in the sub-field on generative modeling. Mainstream generative modeling assumes that there is a phenomena generating the observed data and strive to build a model of that phenomena, which would, for instance, allow generating further observations. Traditional generative modeling considers only *in-distribution* generation where the goal is to generate objects from the category or categories of already observed objects. In terms of novelty generation, this can be considered as generating look-a-likes of known *types* of objects. Although there is considerable value in in-distribution generation (e.g., for super-resolution (Freeman et al., 2002; Dong et al., 2014; Ledig et al., 2016) or in-painting (Xie et al., 2012; Cho, 2013; Yeh et al., 2016)), this perspective is limited from a strict point of view of creativity: it is *unlikely* to come up with a *flying ship* by generating samples from a distribution of *ships* and *flying objects*.

Researchers in creativity research (Runco & Jaeger, 2012) have argued that the crux of creative process is the ability to build new categories based on already known categories. However, creativity is beyond a simple combination exploration: it is about generating previously unknown but meaningful (or valuable) new types of objects using previously acquired knowledge (Hatchuel & Weil, 2009; Kazakçı, 2014). Under this perspective, novelty generation aims at exhibiting an example from a new type. This objective, which we shall call *out-of-distribution generation*, is beyond what can be formalized within the framework of traditional learning theory, even though learning existing types is a crucial part of the process.

From a machine learning point of view, generating an object from an unknown type is not a well-defined problem, and research in generative modeling usually aims at *eliminating* this possibility altogether, as this is seen as a source of instability (Goodfellow et al., 2014; Salimans et al., 2016) leading to spurious samples (Bengio et al., 2013b). In a way, sampling procedures are designed to kill any

possibility of sampling out of the distribution, which is a problem for studying the generation of novelty by machine learning methods.

Arguably, the most important problem is the evaluation of what constitutes a good model for generating out-of-distribution. On the one hand, we are seeking to generate *meaningful* novelty, not trivial noise. On the other hand, we aim at generating *unknown* objects, so traditional metrics based on the concept of likelihood are of no use since novelty in the out-of-distribution sense is unlikely by definition. This lack of metrics hinders answering the first two questions. Without a clear-cut evaluation process, the utility of extending the definition of novelty generation to out-of-sample seems pointless.

This paper argues that for a wider adoption of novelty generation as a topic for scientific study within machine learning, a new engineering principle is needed, which would enable such evaluation, and consequently, rigorous experimental research. In the traditional supervised context, the main engineering design principle is the minimization of the error on a hold-out test set. The paper proposes a simple setup where the generative potential of models can be evaluated by *holding out entire classes*, simulating thus unknown but meaningful novelty. The goal of the novelty generator is then to use training classes to build a model that can generate objects from future (hold-out) classes, unknown at training time.

The main contributions of this paper:

- We design an experimental framework based on hold-out classes to develop and to analyze out-of-distribution generators.
- We review and analyze the most common evaluation techniques from the point of view of measuring out-of-distribution novelty. We argue that likelihood-based techniques inherently limit exploration and novelty generation. We carefully select a couple of measures and demonstrate their applicability for out-of-distribution novelty detection in experiments.
- We run a large-scale experimentation to study the ability of novelty generation of a wide set of different autoencoders and GANs. The goal here is to re-evaluate existing architectures under this new goal in order to open up exploration. Since out-of-distribution novelty generation is arguably a wider (and softer) objective than likelihood-driven sampling from a fixed distribution, existing generative algorithms, designed for this latter goal, constitute a small subset of the algorithms able to generate novelty. The goal is to motivate the reopening some of the closed design questions.

The paper is organized as follows. We review some of the seminal work at the intersection of machine learning and out-of-distribution generation in Section 8.2.

We outline the families of evaluation metrics, focusing on those we use in the paper in Section 8.4. In Section 8.4.3 we describe the gist of our experimental setup needed to understand the metrics described in Section 8.4.4, designed specifically for the out-of-distribution setup. We describe the details of the experimental setup and analyze our results in Section 8.5. Finally, we conclude in Section 8.6.

8.2 Machine learning and novelty generation: the innovation engine, "zero-shot" learning, and discovering new types

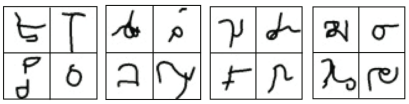
There are three important papers that consider novelty generation in a machine learning context. Nguyen et al. (2015b) proposes an innovation engine (Figure 8.1a). They generate images using a neural net that composes synthetic features. The generator is fed back with an entropy-based score (similar to objectness; Section 8.4.2) coming from an Imagenet classifier, and the feedback is used in an evolutionary optimization loop to drive the generation. An important contribution of the paper is to demonstrate the importance of the objectness score. They show that interesting objects are not generated when asking the machine to generate from a single given class. The generation paths often go through objects from different classes, "stepping stones" which are seemingly unrelated to the final object. The main conceptual difference between our approaches is that Nguyen et al. (2015b) do not ground their generative model in learned knowledge: their generation process is not learned model, rather a stochastic combinatorial engine. On the one hand, this makes the generation (evolutionary optimization) rather slow, and on the other, the resulting objects reflect the style of the (preset) synthetic features rather than features extracted from existing objects.

The main goal of Lake et al. (2015) and Rezende et al. (2016) is *one-shot learning and generation*: learn to classify objects given a small number (often one) of examples coming from a given category, and learn to generate new objects given a single example (Figure 8.1b). One-shot generation is definitely an intermediate step towards out-of-distribution generation. The extremely low number of examples conceptually limits likelihood-based learning/fitting/generation. Lake et al. (2015) circumvents this problem by learning strong Bayesian top-down models (programs) that capture the structural properties of known objects which are generalizable across classes. They also consider unconstrained ("zero-shot") generation as an extension of their approach, and show that the model can generate new symbols from scratch. They make no attempt to conceptualize the goal of unconstrained generation outside the top-down Bayesian framework, or to design evaluation metrics to assess the quality of these objects, but their intriguing results

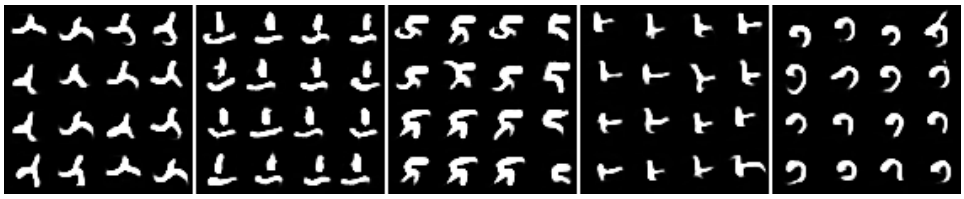
are one of the strongest motivations of our paper.

In (Kazakçı et al., 2016) they show that symbols of new types can be generated by carefully tuned autoencoders, learned entirely bottom-up, without imposing a top-down Bayesian architecture (Figure 8.1c). They also make a first step of defining the conceptual framework of novelty generation by arguing the goal of generating objects from new *types*, unknown at the time of training. They design a technique for finding these new types semi-automatically (combining clustering and human labeling). They argue the importance of defining the *value* of these new types (and of out-of-distribution generation in general), but they make no attempt to design evaluation metrics, thus limiting the exploration and the development of out-of-distribution generative architectures.

(a) "Synthetic" objects from imagenet categories from Figure 7 of (Nguyen et al., 2015b)



(b) "Unconstrained" symbols from Figure 7 of (Lake et al., 2015)



(c) New types of symbols from Figure 6 of (Kazakçı et al., 2016)

Figure 8.1: Examples of generating new objects or types.

8.3 Probabilistic vs. constructive generative models

The generative process is commonly framed in a probabilistic setup: it is assumed that an underlying unknown likelihood *model* $p(\cdot)$ should first be learned on an i.i.d. *training* sample $\mathcal{D} = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, assumed to be generated from $p(\cdot)$, and then a *sampler* $\mathcal{S}$ should sample from the learned $\widehat{p}(\cdot)$. The first step, estimating $p(\cdot)$ using $\mathcal{D}$, is a classical function learning problem that can be studied through the usual concepts of overfitting and regularization, and algorithms can be designed using the classical train/test principle. The second step, designing $\mathcal{S}$ for sampling from $\widehat{p}(\cdot)$ is also a classical domain of random sampling with a conceptual framework and a plethora of methods.

Technically both steps are notoriously hard for the high-dimensional distributions and the complex dependencies we encounter in interesting domains. Hence, most of the recent and successful methods get rid of the two-step procedure at the level of algorithmic design, and short-cut the procedure from the probabilistic $\mathcal{D} \to \mathcal{P} \to \mathcal{S}$ to the constructive $\mathcal{D} \to \mathcal{A}$, where $\mathcal{A}(\mathcal{D})$ is a *generator*, tasked to produce sample objects *similar* to elements of $\mathcal{D}$ but *not identical* to them. $\mathcal{A}$ is fundamentally different from $(\mathcal{P}, \mathcal{S})$ in that there is no explicit fitting of a *function*, we use $\mathcal{D}$ to directly *design an algorithm* or a *program*.

When the probabilistic setup is still kept for *analysis*, we face a fundamental problem: if we assume that we are given the true likelihood function $\mathcal{P}(\cdot)$, the likelihood of the training sample $\frac{1}{n}\sum_{i=1}^{n} \log \mathcal{P}(\mathbf{x}_i)$ is a random variable drawn independently from the distribution of log-likelihoods of i.i.d. samples of size n, so the trivial generator $\mathcal{A}$ which *resamples* $\mathcal{D}$ will have the same expected log-likelihood as an optimal i.i.d. sampler. The resampling "bug" is often referred to as "overfitting". While it makes perfect sense to talk about overfitting in the $\mathcal{D} \to \mathcal{P} \to \mathcal{S}$ paradigm (when $\mathcal{P}$ is *fitted* on $\mathcal{D}$), it is somewhat conceptually misleading when there is no fitting step, we propose to call it "memorizing". When a generator $\mathcal{A}$ is trained on $\mathcal{D}$ without going through the fitting step $\mathcal{D} \to \mathcal{P}$, the classical tools for avoiding memorizing (regularization, the train/test framework) may be either conceptually inadequate or they may not lead to an executable engineering design principle.

The conceptual problem of analyzing constructive algorithms in the probabilistic paradigm is not unrelated to our argument of Section 8.1 that the probabilistic generative framework is too restrictive for studying novelty generation and for designing out-of-distribution generative models. In our view, this flaw is not a minor nuisance which can be fixed by augmenting the likelihood to avoid resampling, rather an inherent property which cannot (or rather, should not) be fixed. The probabilistic framework is designed for generating objects from the distribution of known objects, and this is in an axiomatic contradiction with generating out-of-distribution novelty, objects that are *unknown* at the moment of assembling a training sample. Resampling (generating *exact* copies) is only the most glaring demonstration of a deeper problem which is also present in a more subtle way when attempting to generate new *types* of objects.

We are not arguing that the probabilistic generative framework should be banished, it has a very important role in numerous use cases. Our argument is that it is not adequate for modeling out-of-distribution novelty generation. What follows from this on the *algorithmic* level is not revolutionary: the design of most successful generative algorithms already moved beyond the probabilistic framework. On the other hand, moving beyond the probabilistic generative framework at a *conceptual* level is a paradigm change which will require groundwork for laying the

foundations, including revisiting ideas from a domain larger than machine learning.

At the algorithmic/computational level the machine learning community has already started to move beyond likelihood. The overfitting problem is often solved by implicitly constraining $\mathcal{A}$ not to resample. Another common solution is to design tractable likelihood surrogates that implicitly penalize memorization. These surrogates then can be used at the training phase (to obtain non-resampling generators explicitly) and/or in the evaluation phase (to eliminate generators that resample). The ingenious idea of using discriminators in GANs (Goodfellow et al., 2014; Salimans et al., 2016) is a concrete example; although the setup *can* be analyzed through the lens of probabilistic sampling, one does not *have to* fall back onto this framework. If we drop the underlying conceptual *probabilistic* framework, the *constructive* GAN idea may be extended beyond generating from the *set* which is indistinguishable from the set of existing objects. In Section 8.4.4 we will use discriminators to assess the quality of generators whose very goal is to generate novelty: objects that *are* distinguishable from existing objects. The main challenge is to avoid the trivial novelty generator, producing uninteresting noise. This challenge is structurally similar to avoiding the trivial memorizing/resampling generator in in-distribution sampling. The two main elements that contribute to the solution is i) to ground the generator strongly in the structure of existing *knowledge*, without overly fixating it on existing *classes*, and ii) use a discriminator which knows about *out-of-class* novelty to steer architectures towards novelty generation.

8.4 Evaluation of generative models

In this section we outline the families of evaluation metrics, focusing on those we use in the paper. In Section 8.4.3 we describe the gist of our experimental setup needed to understand the metrics described in Section 8.4.4, designed specifically for the out-of-distribution setup.

8.4.1 Indirect supervised metrics

When generative models are used as part of a pipeline with a supervised goal, the evaluation is based on the evaluation of the full pipeline. Examples include unsupervised pre-training ((Hinton et al., 2006; Bengio et al., 2007); the original goal that reinvigorated research in neural nets), semi-supervised learning (Kingma et al., 2014; Rasmus et al., 2015; Maaløe et al., 2016; Salimans et al., 2016), inpainting (Xie et al., 2012; Cho, 2013; Yeh et al., 2016), or super-resolution (Freeman et al., 2002; Dong et al., 2014; Ledig et al., 2016). The design goal becomes straightforward, but the setup is restricted to improving the particular pipeline, and

there is no guarantee that those objectives can be transferred between tasks. In our case, the objective of the supervised pipeline may actually suppress novelty. In a certain sense, GANs also fall into this category: the design goal of the generator is to fool a high-quality discriminator, so the generator is asked *not* to generate new objects which can be easily discriminated from known objects. In our experiments, surprisingly, we found that GANs can be still tuned to generate out-of-distribution novelty, probably due to the deficiencies of both the generator and the discriminator. Our goal in this paper can also be understood as designing a pipeline that turns novelty generation into a supervised task: that of generating objects from classes unknown at training time.

Parzen density estimator

Parzen density estimators are regularly used for estimating the log-likelihood of a model (Breuleux et al., 2009). A kernel density estimator is fit to generated points, and the model is scored by log-likelihood of a hold-out test set under the kernel density. The metrics can be easily fooled (Theis et al., 2015), nevertheless, we adopted it in this paper for measuring both the in-distribution and out-of-distributions quality of our generators.

8.4.2 Objectness

Salimans et al. (2016) proposed a new entropy-based metrics to measure the "objectness"[1] of the generated *set* of objects. As GANs, the metrics uses a trained discriminator, but unlike GANs, it is not trained for separating real objects and generated objects, rather to classify real objects into existing categories. The goal of the generator is create objects which belong confidently to a low number (typically one) of classes. To penalize generators fixating onto single objects or categories, they also require that the *set* of objects has a high entropy (different objects span the space of the categories represented by the discriminator). The metrics is only indirectly related to classical log-likelihood: in a sense we measure how likely the objects are *through the "eye" of a discriminator*.

Formally, objectness is defined as

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{\ell=1}^{K} p_{i,\ell}\log\frac{p_{i,\ell}}{p_\ell},$$

[1] They also call it "inception score" but we found the term objectness better as it is more general than the single model used in their paper.

where K is the number of classes,

$$p_{i,\ell} = \mathcal{P}(\ell|\mathbf{x}_i)$$

is the posterior probability of category ℓ given the generated object $\mathbf{x}_i$, under the discriminator $\mathcal{P}$ trained on a set with known labels, and

$$p_\ell = \frac{1}{n}\sum_{i=1}^{n} p_{i,\ell},$$

are the class marginals.

Salimans et al. (2016) proposed objectness as one of the "tricks" to stabilize GANs, but, interestingly, a similar measure was also used in the context if evolutionary novelty generation (Nguyen et al., 2015b).

8.4.3 Assessing out-of-distribution novelty by out-of-class scoring

As the classical supervised validation setup simulates past (training) and future (test) by randomly partitioning an existing data set, we can simulate existing knowledge and novelty by partitioning existing data sets *holding out entire classes*. The goal of the novelty generator is then to use training classes to build a model that can generate objects from future (hold-out) classes, unknown at training. In our first experiments we tried to leave out single classes of MNIST, but the label noise "leaked" hold-out classes which made the evaluation tricky. To avoid this, we decided to challenge the generator, trained on MNIST, to generate *letters*. We pre-trained various discriminators using different setups, only on digits (MNIST), only on letters (Google fonts, a total of 1268 fonts which gave a total of 32968 examples), or on a mixture of digits and letters, and used these discriminators to evaluate novelty generators in different ways. For example, we measure *in-class objectness* and *in-class Parzen* using a discriminator trained on MNIST, and *out-of-class objectness* and *out-of-class Parzen* by a discriminator trained on (only) Google fonts.

8.4.4 Out-of-class scores

Naturally, letter discriminators see letters everywhere. Since letters are all they know, they classify everything into one of the letter classes, quite confidently (this "blind spot" phenomenon is exploited by (Nguyen et al., 2015b) for generating "synthetic" novelty), the letter objectness of an in-distribution digit generator can sometimes be high. For example, a lot of *6*s were classified as *b*s. To avoid this "bias", we also trained a discriminator on the union of digits and letters, allowing it to choose digits when it felt that the generated object looked more like a digit.

We designed two metrics using this discriminator: *out-of-class count* measures the frequency of confidently classified letters in a generated set, and *out-of-class max* is the mean (over the set) of the probability of the most likely letter. None of these metrics penalize "fixated" generators, outputting the same few letters all the time, so we combine both metrics with the entropy of the letter posterior (conditioned on being a letter).

Formally, let $p_{i,1}, \ldots, p_{i,K_{\text{in}}}$ be the in-class posteriors and $p_{i,K_{\text{in}}+1}, \ldots, p_{i,K_{\text{in}}+K_{\text{out}}}$ be the out-of-class posteriors, where $K_{\text{in}} = 10$ is the number of in-class classes (digits), and $K_{\text{out}} = 26$ is the number of out-of-class classes (letters). Let

$$\ell_i^* = \arg\max_{\ell} p_{i,\ell}$$

and

$$\ell_{\text{out}\,i}^* = \arg\max_{K_{\text{in}}<\ell\leq K_{\text{in}}+K_{\text{out}}} p_{i,\ell}$$

be the most likely category overall and most likely out-of-class category, respectively. Let

$$\tilde{p}_\ell = \frac{\sum_{i=1}^n \mathbb{I}\left\{\ell = \ell_{\text{out}\,i}^*\right\}}{\sum_{i=1}^n \mathbb{I}\left\{\ell_{\text{out}\,i}^* > K_{\text{in}}\right\}}$$

be the normalized empirical frequency of the out-of-class category ℓ. We measure the diversity of the generated sample by the normalized entropy of the empirical frequencies

$$\text{diversity} = -\frac{1}{\log K_{\text{out}}} \sum_{\ell=K_{\text{in}}}^{K_{\text{in}}+K_{\text{out}}} \tilde{p}_\ell \log \tilde{p}_\ell,$$

and define:

$$\begin{aligned}\text{out-of-class-count} =& (1-\lambda) \times \frac{1}{n}\sum_{i=1}^n \mathbb{I}\left\{\ell_i^* > K_{\text{in}} \wedge p_{i,\ell_i^*} > \theta\right\} + \\ & \lambda \times \text{diversity},\end{aligned} \tag{8.1}$$

and

$$\text{out-of-class max} = (1-\lambda) \times \frac{1}{n}\sum_{i=1}^n p_{i,\ell_{\text{out}\,i}^*} + \lambda \times \text{diversity}.$$

In our experiments we set the confidence level $\theta = 0.95$ and the mixture coefficient $\lambda = 0.5$.

8.4.5 Human refereeing and the visual Turing test

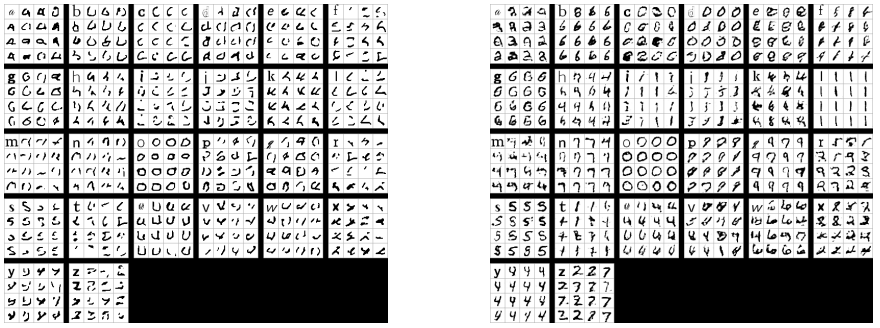
The ultimate test of l'art pour l'art generative models is whether humans like the generated objects. Visual inspection is often used as an evaluation principle in papers (Denton et al., 2015; Radford et al., 2015; Dosovitskiy et al., 2016), and it is sometimes even made part of the objectified pipeline by using crowdsourcing tools (Denton et al., 2015; Lake et al., 2015; Salimans et al., 2016). First, it definitely makes development (e.g., model selection and hyperparameter tuning) slow. Second, the results depend a lot on what questions are asked and how the responders are primed. For testing generative models, the usual GAN-type question to ask is whether the generated objects are generated by a nature (or a human) or a machine (the visual Turing test). Even those that go the furthest in tasking machines to generate novelty (Lake et al., 2015) ask human judges to differentiate between human and machine. In our view, this question is too restrictive when the goal is out-of-distribution novelty generation. Asking whether an object is "new" is arguably too vague, but inventing adjective categories (such as "surprising" or "interesting" (Schmidhuber, 2009)) that can poll our ability to detect novelty should be on the research agenda. Priming is another important issue: the answer of a human annotator can depend on the information given to her. Nevertheless, a human annotation tool with well-designed priming and questions could accelerate research in novelty generation in the same way labeling tools and standard labeled benchmark sets accelerated supervised learning.

We assessed the visual quality of the set of generated objects using an in-house annotation tool. We took each model which appeared in the top ten by any of the quantitative metrics described in the previous section, and hand-labeled them into one of the following three categories: i) letters, ii) digits, and iii) bad sample (noise or not-a-symbol).

Each panel consisted 26×15 generated objects, the fifteen most probable symbols of each letter according to the classifier trained on both letters and digits (Figure 8.2). The goal of this annotation exercise was i) to assess the visual quality of the generated symbols and ii) to assess the quality of the metrics in evaluating novelty.

8.5 Experiments

Our scores cannot be directly optimized because they all measure out-of-class performance, and showing out-of-class objects at training would be "cheating". All our (about 1000) models were trained for "classical" objectives: reconstruction error in the case of autoencoders, and adversarial error in the case of GANs. The out-of-class scores were used as a weak feedback for model selection and (quasi



(a) The top autoencoder (b) The top GAN

Figure 8.2: A couple of the top models according to human assessment. Top left characters of each 4×4 panel are the labels, letters coming from the training sample. For each letter we display the fifteen most probable symbols according to the classifier trained on both letters and digits.

random) hyperparameter optimization. The goal is not to be statistically flawless, after all we do not have a statistical model. Rather we set our goal to analyze existing generative architectures from the point of view of novelty generation. Most of the generative models come from a large class of architectures, sometimes purposefully designed for not to "misbehave". When possible, we turned these tricks, designed to avoid generating "spurious" objects, into optional hyperparameters.

8.5.1 Detailed experimental setup

We used two families of deep learning based generative models, autoencoders and GANs. The architectures and the optional features are described in the next sections. All hyperparameters were selected randomly using reasonable priors. All the $\sim$1000 autoencoders were trained on MNIST training data.

Autoencoder architectures and generation procedure

We used three regularization strategies for autoencoders: sparse autoencoders (Makhzani & Frey, 2013, 2015), denoising autoencoders (Bengio et al., 2013b) and contractive autoencoders (Rifai et al., 2011).

Sparse autoencoders can either be fully connected or convolutional. For fully connected sparse autoencoders, we use the k-sparse formulation from (Makhzani & Frey, 2013), a simple way of obtaining a sparse representation by sorting hidden units and keeping only the top $k\%$, zeroing out the others, and then backpropagating only through non-zero hidden units.

For convolutional sparse architectures, we use the "winner take all" (WTA) formulation from (Makhzani & Frey, 2015) which obtains *spatial sparsity* in convolutional feature maps by keeping only the maximum activation of each feature map, zeroing out the others. We optionally combine it with *channel sparsity* which, for each position in the feature maps, keeps only the maximum activation across the channels and zero out the others.

For contractive autoencoders, we use the fully connected version with a single hidden layer from (Rifai et al., 2011).

We also explore mixtures between the different autoencoder variants in the hyperparameter search. For each model we choose to enable or disable independently the denoising training procedure, the contractive criterion (parametrized by the contractive coefficient, see (Rifai et al., 2011)) and the sparsity rate k (only for fully connected architectures). Table 8.1 shows the hyperparameters and their priors.

The generation procedure we use for autoencoders is based on (Bengio et al., 2013b), who proposed a probabilistic interpretation of denoising autoencoders and a way to sample from them using a Markov chain. To have a convergent procedure and to obtain fixed points, we chose to use a deterministic generation procedure instead of a Markov chain (Bahdanau & Jaeger, 2014). As in (Bahdanau & Jaeger, 2014), we found that the procedure converged quickly.

In initial experiments we found that 100 iterations were sufficient for the majority of models to have convergence so we chose to fix the maximum number of iterations to 100. We also chose to extend the procedure of (Bahdanau & Jaeger, 2014) by binarizing (using a threshold) the images after each reconstruction step, as we found that it improved the speed of the convergence and could lead to final samples with an exact zero reconstruction error. In Figure 8.3, we can see an visualization of the generation procedure of independent samples using the model in Figure 8.5.

For stochastic gradient optimization of the autoencoder models, we used adadelta (Zeiler, 2012) with a learning rate of 0.1 and a batch size of 128. We used rectified linear units as an activation function for hidden layers in all models. We use the sigmoid activation function for output layers.

Generative adversarial networks (GANs)

In our experiments, we built upon (Radford et al., 2015) and used their architecture as a basis for hyperparameter search. We modified the code proposed here to sample new combinations of hyperparameters. Table 8.2 shows the hyperparameters and their priors.

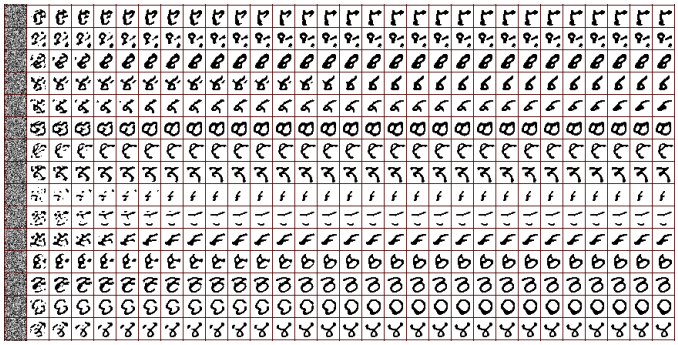


Figure 8.3: A visualization of the iterative procedure of generation from autoencoders using the model in Figure 8.5. Each row is an independent sample and columns are iterations.

Table 8.1: Autoencoder hyperparameter priors.

Name	Prior	Type
nb. layers	1, 2, 3, 4, 5	choice
nb. fc.[2] hidden units	100,200,300,...1000	choice
nb. conv layers	1, 2, 3, 4, 5	choice
nb. conv filters	2^i, i=3...9	choice
conv layers filter size	3 or 5	choice
noise corruption	[0, 0.5]	uniform
k sparsity rate	[0, 1]	uniform
contraction coefficient	[0, 100]	uniform

Name	Prior	Type
nb. discr. updates	1, 2, 3	choice
l2 coeficient	$[10^{-6}, 10^{-1}]$	logspace
gen. input dim.	10...200	choice
nb. fc. gen. units	2^i, i=3...11	choice
nb. fc. discr. units	2^i, i=3...11	choice
nb. filters gen.	2^i, i=3...9	choice
nb. filters discr.	2^i, i=3...9	choice
learning rate	$[10^{-6}, 10^{-1}]$ or 0.0002	logspace
weight initialization	std $\in [10^{-3}, 10^{-1}]$	logspace

Table 8.2: GAN hyperparameter priors.

Discrminators

To compute in-class and out-of-class scores(see Section 8.4), we trained 3 discriminators. The first was trained to classify between digits, the second to classify between letters, and the third to classify between a mixture of digits and letters (by concatenating the digits and letters dataset).

The architecture used in all the discriminators was a convolutional neural network with 3 convolutional layers, each one followed by max pooling. The number of filters for the 3 convolutional layers were respectively 32, 64, and 128. We used filters of size 3. The last convolutional layer was followed by 3 fully connected layers with respectively 512, 256, and 128 hidden units. We used the PReLU (He et al., 2015) activation function everywhere and dropout with $p = 0.5$ in every convolutional and fully connected layer.

8.5.2 Analysis

First, we found that tuning (selecting) generative models for in-distribution generation will make them "memorize" the classes they are trained to sample from, as we can see in Figure 8.4. This is of course not surprising, but it is important to note because it means that out-of-class generation is non-trivial, and the vast majority of architectures designed and tuned in the literature are not generating out-of-class novelty naturally. Second, we did succeed to find architectures and hyperparameter combinations which lead to out-of-class novelty. Most of the generated objects, of course, were neither digits nor letters (Figure 8.5), which is why we needed the "supervising" discriminators to find letter-like objects among them. The point is not that *all* new symbols are letters, that would arguably be an impossible task, but to demonstrate that by opening up the range of generated objects, we do not generate noise, rather objects that *can be* forming new categories.

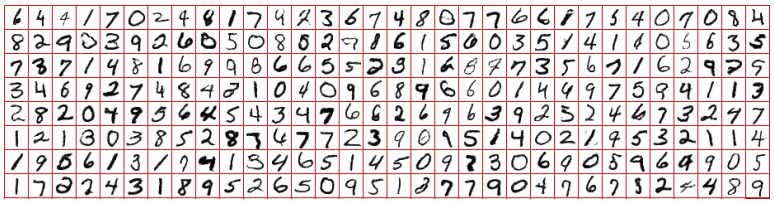


Figure 8.4: A random selection of symbols generated by one of our best models (a GAN) according to in-class scores

The quantitative goal of this study was to assess the quality of the defined *metrics* in evaluating out-of-distribution generators. We proceeded in the following way. We selected the top ten autoencoders and GANs according to the five metrics

Figure 8.5: A random selection of symbols generated by one of our best models (a sparse autoencoder) according to out-of-class scores, the same as the one that generated the letters in Figure 8.6(b).

	inter-score correlations								human counts		
	oc	om	oo	op	ic	im	io	ip	out	in	bad
out count	1	-0.03	-0.13	0.04	-0.12	0.02	-0.07	-0.11	12	0	8
out max	-0.03	1	-0.07	0.01	-0.16	-0.10	0.03	-0.09	15	0	5
out objectness	-0.13	-0.07	1	0.21	-0.06	0.08	0.02	-0.08	9	10	1
out Parzen	0.04	0.01	0.21	1	-0.17	0.01	-0.19	-0.20	4	13	3
in count	-0.12	-0.16	-0.06	-0.17	1	0.30	0.1	0.14	-	-	-
in max	0.02	-0.10	0.08	0.01	0.30	1	0.03	0.06	-	-	-
in objectness	-0.07	0.03	0.02	-0.19	0.1	0.03	1	0.00	-	-	-
in Parzen	-0.11	-0.09	-0.08	-0.20	0.14	0.06	0.00	1	0	17	3

Table 8.3: Inter-score correlations among top 10% models per score and human annotation counts among top twenty models per score. out=letters; in=digits.

of out-of-class (letters) count, out-of-class max, out-of-class objectness, out-of-class Parzen, and in-class Parzen. We then annotated these models into one of the three categories of "letter" (out), "digit" (in), and "bad" (noise or not-a-symbol). The last three columns of Table 8.3 show that the out-of-class count and out-of-class max scores work well in selecting good out-of-class generators, especially with respect to in-class generators. They are relatively bad in selecting good generators overall. Symmetrically, out-of-class objectness and the Parzen measures select, with high accuracy, good quality models, but they mix out-of-class and in-class generators (digits and letters). Parzen scores are especially bad at picking good out-of-class generators. Somewhat surprisingly, even out-of-class Parzen is picking digits, probably because in-distribution digit generators generate more regular, less noisy images than out-of-class letter generators. In other words, opening the space towards non-digit like "spurious" symbols come at a price of generating less clean symbols which are farther from letters (in a Parzen sense) than clean digits.

We also computed the inter-score correlations in the following way. We first selected the top 10% models for each score because we were after the correlation of the best-performing models . Then we computed the Spearman rank correlation of the scores (so we did not have to deal with different scales and distributions). The first eight columns of Table 8.3 show that i) in-class and out-of-class measures are anti-correlated, ii) out-of-class count and max are uncorrelated, and are somewhat anti-correlated with out-of-class objectness.

These results suggest that the best strategy is to use out-of-class objectness for selecting good quality models and out-of-class count and max to select models which generate letters. Figure 8.6 illustrates the results by pangrams (sentences containing all letters) written using the generated symbols. The models (a)-(d) were selected automatically: these were the four models that appeared in the top ten both according to out-of-class objectness and out-of-class counts. Letters of the last sentence (e) were hand-picked by us from letters generated by several top models. Among the four models, three were fully connected autoencoders with sparsity and one was a GAN. All of the three sparse autoencoders had five hidden layers and used a small noise corruption (less than 0.1). The GAN used the default learning rate of 0.0002 and a large number (2048) of fully connected hidden units for the generator, while the number of fully connected hidden units of the discriminator was significantly smaller (128).

(a) (b) (c) (d) (e)

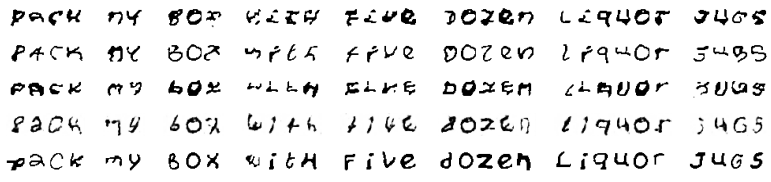


Figure 8.6: Pangrams created (a-d) using top models selected automatically, and (e) using letters selected from several models by a human.

8.6 Conclusion

In this chapter we have proposed a framework for designing and analysing generative models for novelty generation. The quantitative measures make it possible to systematically study the creative capacity of generative models. We believe that human evaluation will remain an important source of feedback in this domain for the foreseeable future. Nevertheless, quantitative measures, such as our out-of-class objectness and out-of-class count and max, will i) make it possible to semi-automate the search for models that exhibit creativity, and ii) allow us to study,

from the point of view of novelty generation, the numerous surrogates used for evaluating generative models (Theis et al., 2015), especially those that explicitly aim at quantifying creativity or interestingness (Schmidhuber, 2009).

The main focus of this chapter was setting up the experimental pipeline and to analyze various quality *metrics*, designed to measure out-of-distribution novelty of samples and generative models. The immediate next goal is to analyze the *models* in a systematic way, to understand what makes them "memorizing" classes and what makes them opening up to generate valuable out-of-distribution samples.

Chapter 9

De novo drug design with deep generative models: an empirical study

In this chapter, we propose to use an external value function to judge the capacity of models to generate useful novelty in a real setting. The main goal of this chapter is to demonstrate the usefulness of novelty generation in a scientific task. We present an empirical study about the using RNN generative models for stochastic optimization in the context of de novo drug design. We study different kinds of architectures and we find models that can generate molecules with higher values than ones seen in the training set. Our results suggest that we can improve traditional stochastic optimizers, that rely on random perturbations or random sampling by using generative models trained on unlabeled data, to perform knowledge-driven optimization. The content of this chapter is from the paper Cherti, Kégl, and Kazakçı (2017).

9.1 Introduction

The goal of computer-based *de novo* drug design is to build new molecules from scratch that can be used as drugs. The molecules are designed in a way that it will bind to a target (for instance to a human protein or to a virus) to change its behavior. When a molecule binds to the desired target, it is called a *ligand*.

Most available molecule generation techniques rely on combination of a library of 'fragments'. A fragment is subgraph of a molecule structure, it can be an atom or a group of atoms (e.g., a ring). The fragments are combined in a chemically meaningful way to obtain new molecules (Schneider & Schneider, 2016). One advantage of using fragments is to reduce the search space, which would be huge if molecule

structure generation was done one atom at a time, like it was done in atom-based structure generation techniques which are now less popular (Hartenfeller & Schneider, 2011; Schneider, 2013). Another advantage is that the fragments are extracted manually from known drug molecules and from chemistry knowledge. On the other hand as the library of fragments is fixed, it can constrain the search space too much and thus it might be possible to overlook some interesting molecules.

There are two main techniques for molecule generation depending on how much information we have about the target: receptor-based and ligand-based techniques. *Receptor-based* techniques are used when the 3D structure of the target is available, while *ligand-based* techniques are used when a set of molecules that bind to a given target are already known and the goal is to find new molecules that can bind to the target even better or that can satisfy other constraints like ease of synthesis.The ligand-based algorithms take as input a set of ligands that are known to bind to a target, and return a new set of ligands that bind to the target.

Most approaches in the literature involve discrete optimization techniques like genetic algorithms operating on the graph representation of the molecules.We point to the reader some works about techniques used traditionally to design new molecules: Gillet et al. (1993); Wang et al. (2000); Pierce et al. (2004); Douguet et al. (2005); Fechner & Schneider (2006); Dey & Caflisch (2008); Kutchukian & Shakhnovich (2010); White & Wilson (2010); Li et al. (2011); Hartenfeller et al. (2012); Li et al. (2016); Masek et al. (2016).

As pointed out by Gómez-Bombarelli et al. (2016b), current approaches rely on handcrafted rules for perturbing or hybridizing molecules or inserting fragments to obtain new molecules. While these rules rely on chemistry knowledge, they may bias the search space towards a specific portion of the space and some interesting molecules might be overlooked. One attempt to solve this issue would be to replace the handcrafted rules by rules learned from data, using generative models.

9.2 Experiments

9.2.1 Methodology and objective

As in Gómez-Bombarelli et al. (2016a); Segler et al. (2017), we used the SMILES (Weininger, 1988) representation of molecules, which is a textual representation of the molecular graph using a formal language. Like in Gómez-Bombarelli et al. (2016a); Segler et al. (2017), we used recurrent neural networks (RNNs) to learn a generative model of sequences for SMILES strings from a dataset of known molecules.

We used the same dataset than Gómez-Bombarelli et al. (2016a), which is a subset of the ZINC dataset (Irwin et al., 2012; Sterling & Irwin, 2015).We split the dataset used in Gómez-Bombarelli et al. (2016a) into a training and a validation

set. The size of the training set was 200000, while the size of the validation set was about 50000. We sampled the hyperparameters of the models randomly from a prior (see the appendix for more details) then we trained the models on the training set and used the validation set for early stopping. Finally, for each molecule we sampled from the model we computed the score used in Gómez-Bombarelli et al. (2016a):

$$J(m) = \text{LogP}(m) - \text{SA}(m) - \text{ring-penalty}(m) \tag{9.1}$$

A high $J(m)$ selects drug-like molecules, as measured by $LogP$(the partition coeficient (Wildman & Crippen, 1999)) and penalizes molecules which are difficult to synthesize, measured by SA (synthetic accessibility) and ring-penalty. See the appendix for more details.

This score assesses each molecule, rather than a set of molecules generated by a model. To evaluate the models, we use the expected improvement criterion (EI; Močkus (1975)), used routinely in Bayesian optimization (Jones et al., 1998), to assess how much the score $J(m)$ improved over the training set. Given a set of M generated molecules $M = \{m_1, m_2, ..., m_M\}$ and the maximum score $J(m)$ on the training set $D = \{x_1, x_2, ..., x_N\}$, $J^*(D) = max_{i=1...N} J(x_i)$, the expected improvement(EI) is defined as:

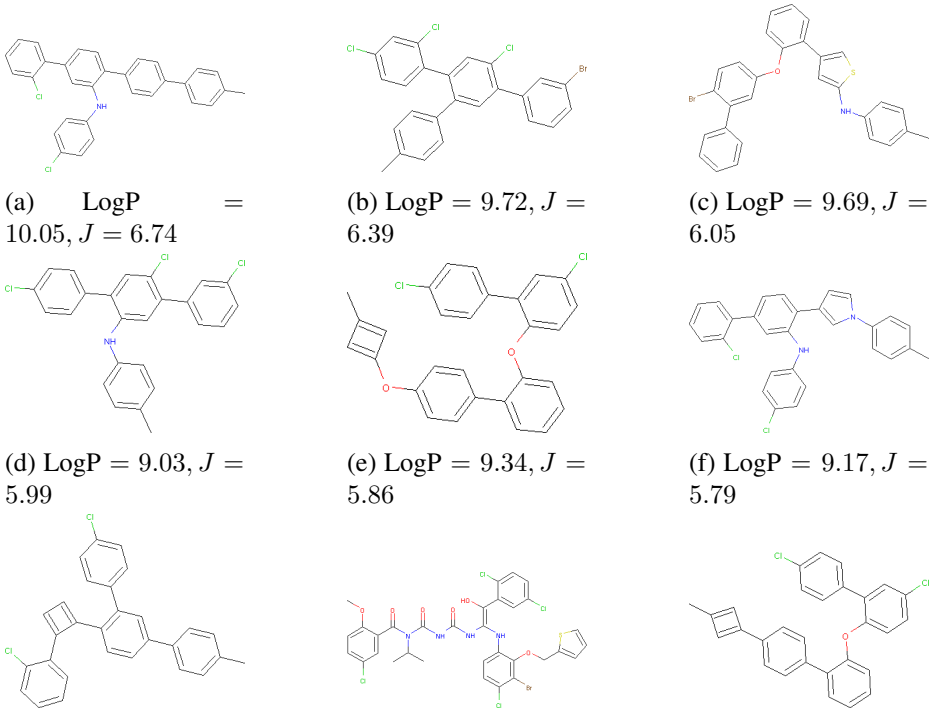
$$EI(M) = \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}\{J(m_i) > J^*(D)\}(J(m_i) - J^*(D)), m_i \in M^1 \tag{9.2}$$

9.2.2 Models and results

We used two kinds of RNN architectures. The first corresponds to char-RNNs (Karpathy et al., 2015) where the RNN is trained to predict the next character based on the full history of previous characters. The second type is a sequence-to-sequence autoencoder, analog to Sutskever et al. (2014), but where the input and the output are the same and we do not use teacher forcing (Williams & Zipser, 1989).

We trained a total of 480 models. Among the top 8 of our models according to EI(eq. 9.2), 4 were autoencoders with a convolutional encoder and an RNN decoder, the remaining 4 models were autoencoders with an RNN encoder and decoder. Noise turned out to be helpful: all of these models were using a denoising criterion (Bengio et al., 2013b) (during training and generation) which we adapted for sequential data (see the noise procedure we use in the appendix). None of the

[1] $\mathbb{I}\{x > y\}$ is the indicator function, 1 if $x > y$, 0 if not.



(a) LogP $= 10.05, J = 6.74$

(b) LogP $= 9.72, J = 6.39$

(c) LogP $= 9.69, J = 6.05$

(d) LogP $= 9.03, J = 5.99$

(e) LogP $= 9.34, J = 5.86$

(f) LogP $= 9.17, J = 5.79$

(g) LogP $= 9.17, J = 5.79$

(h) LogP $= 9.98, J = 4.91$

(i) LogP $= 9.46, J = 6.05$

Figure 9.1: Generated molecules which have $LogP$ and J(eq.9.1) better than training data D, for which $LogP^*(D) = 8.25$ and $J^*(D) = 4.52$. We note that except for 9.1h, all the other molecules have a better LogP and J than the ones in Gómez-Bombarelli et al. (2016a), for which the best reported molecule had LogP $= 8.51$ and $J = 5.02$.

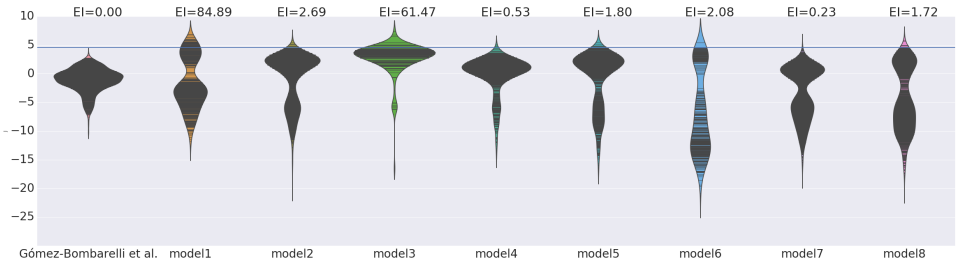


Figure 9.2: Violin plots showing the distributions of J(eq.9.1) of the molecules generated by the top 8 of our models and the ones generated by the model proposed in Gómez-Bombarelli et al. (2016a). The blue horizontal line corresponds to the highest value of J in the training data, $J^*(D) = 4.52$. We report the expected improvement $EI * 10^3$(eq.9.2) of each model. We can see that our models exhibit more diversity than the baseline (Gómez-Bombarelli et al., 2016a), this allows them to find molecules with better scores.

autoencoders were using LSTMs, they were either using GRUs (Cho et al., 2014) or vanilla RNNs (Karpathy et al., 2015).

To compare the models, we generated 100K molecules from each model (among the top 8 models), including the baseline variational autoencoder of Gómez-Bombarelli et al. (2016a)[2]. We discarded illegal and duplicate molecules. The distribution of the score J (eq.9.1) is visualized by violin plots in Figure 9.2. Interestingly, our models do not necessarily generate better molecules *on average*, but the thicker upper tail of the distributions ensures that there is significant mass above the best molecules in the training set. In a stochastic optimization loop generating a lot of suboptimal candidates is not necessarily a problem since the selection operator can get rid of them. On the other hand, generating a significant number of high-value candidates (above the current best) accelerates the optimization.

9.3 Details about the experiments

9.3.1 Preprocessing

We preprocessed the SMILES strings as the following. A special begin character and end character are added respectively in the beginning and the end of the strings. As in Gómez-Bombarelli et al. (2016b), the maximum length of the strings were

[2]We sampled from their variational autoencoder by first sampling from the gaussian prior on the latent space then decoded back to the string space stochastically using the decoder.We used the code provided here.

120, we thus padded all the strings after the end character with a special zero character so that each string had a length of 120. We converted the strings into onehot representation which was the input to the models.

9.3.2 Objective function

The objective function we optimize is:

$$J(m) = \text{LogP}(m) - \text{SA}(m) - \text{ring-penalty}(m) \tag{9.3}$$

where LogP is the partition coefficient (Wildman & Crippen, 1999)[3], SA is the synthetic accessibility (Ertl & Schuffenhauer, 2009)[4] and ring-penalty is defined as: ring-penalty(m) = max-ring-length(m) if max-ring-length$(m) > 6$ else 0 [5], where max-ring-length(m) of a molecule is the maximum ring length among the rings contained in the molecule. We standardize each term ($LogP$, SA and ring-penalty) by subtracting the mean and dividing by the standard deviation (both computed on the training set).

9.3.3 Models

We used two kinds of architectures, Char-RNNs and sequential autoencoders. We provide below details about the architectures, the prior of the hyperparameters and how the generation is done. To train a new model, we sample from the prior of the hyperparameters and we train the model. After the model is trained, we generate samples from it, filter the duplicates and the non valid strings(using RDKIT (Landrum, 2006)), then compute the scores provided in eq.9.3 (using RDKIT (Landrum, 2006)) and eq.9.2. We used keras (Chollet et al., 2015) in our experiments.

Char-RNNs

As in Karpathy et al. (2015), we used a stack of RNN layers that predict the character at time step t then feed back in the next time step. The number of layers were between 1 and 5. We used a dropout rate in each layer, where the dropout rate could be either 0 (disabled) or 0.1 or 0.3 or 0.5. The size of the hidden state in each layer was selected from $\{100, 200, 300, ..., 1000\}$. The type of the parametrization of the RNN could either be a Vanilla RNN (Karpathy et al., 2015), a GRU (Cho et al., 2014) or an LSTM (Hochreiter & Schmidhuber, 1997).

[3] We used RDKIT (Landrum, 2006) to compute the partition coefficient.
[4] We used the code here from RDKIT (Landrum, 2006) to calculate synthetic accessibility.
[5] Personal communication with Rafael Gómez-Bombarelli and José Miguel Hernández-Lobato.

For generation, we initialize the string by the begin character, predict the next character probabilities, sample then feed back the sampled character as an input to the next timestep. We repeat the process until the end character is generated.

Sequential autoencoders

We used sequential autoencoders as in Bowman et al. (2015); Gómez-Bombarelli et al. (2016a) but without the variational objective and without teacher forcing (Williams & Zipser, 1989; Lamb et al., 2016). The advantage of teacher forcing is that it makes training much easier, on the other hand, it makes the model rely less on the fixed-length representation computed from the input sequence and more on the input tokens fed to the decoder at each time step. The encoder was either a stack of 1d convolutions like Gómez-Bombarelli et al. (2016a) or a stack of RNNs.

For convolutional encoders, the number of filters and the size of the filters were respectively selected from $\{8, 16, 32, 64, 128\}$ and $\{3, 5, 7, 9, 11\}$. The number of encoder layers was between 1 and 5. We used a fully connected layer with a linear activation after the last convolution. The number of hidden units in that layer was selected from $\{100, 200, 300, ..., 1000\}$. The fully connected layer was used to condition the decoder, which was a stack of RNN layers where the number of hidden layers was between 1 and 5. The size of the hidden state of the decoder layers was selected from $\{100, 200, 300, ..., 1000\}$. We used the 'relu' activation in all convolutional layers.

For RNN encoders, the number of encoder layers were between 1 and 5. The size of the hidden state in each layer was selected from $\{100, 200, 300, ..., 1000\}$. We use the hidden state in the last time step of the last RNN of the encoder to condition the decoder. The decoder part was identical to sequential autoencoders with a convolutional encoder, defined above.

Sequential autoencoders were trained to reconstruct the input sequence from a noisified version of the input, this makes them a kind of denoising autoencoders (Vincent et al., 2008).

Noise was applied for each input character independently: with a probability p the i-th character was replaced with a random character uniformly from the vocabulary. Noise probability was selected from $\{0, 0.1, 0.2, 0.3, 0.5\}$.

For generation, inspired by Bengio et al. (2013b), we use an iterative generation procedure. We start by a completely random string. In each iteration, we apply the same noise procedure we used during training (with the same noise probability), then we reconstruct and feed the reconstructed input again to the autoencoder. We repeat this process for 100 iterations and save all the generated sequences along the chain. The advantage we see with this kind of generation is that contrary to Char-RNNs or variational autoencoders where the generation is one-shot, this generation

procedure allows a form of iterative refinement of the string because the repeated application of the autoencoder reconstruction function goes towards regions with high probability density (Bengio et al., 2013b; Alain & Bengio, 2014).

9.4 Conclusion

In this chapter, we proposed an experimental study where we use recurrent neural networks (RNNs) to generate molecules in the context of drug design. We experimented with different kinds of RNNs and found ones that can generate molecules that score higher than the ones that are available in the training set, according to a domain specific evaluation criterion. The results are encouraging, and suggest a potential use of such models to improve traditional stochastic optimizers by learning a model from data and then use it to generate good candidates. This work was a first step. In the future, it would be interesting to extend this work by making the generative process of the RNNs aware of the evaluation criterion, as currently the generation and the evaluation are two separated steps.

Chapter 10

The characteristics of models that can generate novelty

In this chapter, the goal is to explain and characterize the models that can generate novelty, such as the ones we found empirically in Chapter 8. Since we found models that could generate novelty (according to the definition and the scores used in Chapter 8) as well as models that could generate only known objects, it is natural to ask how those two kinds of models differ. In this chapter, we will show that when the capacity of the models is restricted through different mechanisms such as sparsity or noise or by limiting the size of the bottleneck, the model is less able to generate novelty, and is encouraged to generate known objects only.

In Section 10.1, we will describe the model family which will study in this chapter, and we will visualize the learned features in Section 10.2. In Section 10.3, we argue that for a model to generate novelty it should at least be able to *represent* not only examples from the known classes but also hopefully examples from unseen classes which would use the same features learned on the known classes. To study this, we introduce a score to measure the capability of a model to recognize examples from unseen classes. In section 10.4, we show that by restricting the capacity of the models, their ability to recognize examples from unseen classes decrease, which also intuitively suggest that the models which cannot recognize examples from unseen classes would not be able to *generate* novelty, as they are unable to represent them. In Section 10.5 and Section 10.6, we confirm that the models that are good at generating novelty, such as the ones we found in Chapter 8, are also good at recognizing images from unseen classes, and restricting the capacity of the models reduce the ability of those models to generate novelty.

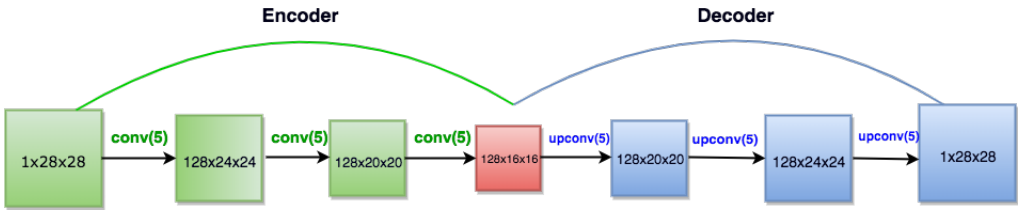


Figure 10.1: An example architecture from the convolutional autoencoder model family which we study in this section, with $L = 3$. Each square represents a layer and refers to a volume of shape $(\text{depth}, \text{height}, \text{width})$, denoted in the figure as *depth* x *height* x *width*. Here the input layer is of size 28x28 and have 1 color channel. Each encoder layer has filters of size 5 and reduce the feature maps width and height size by 4. Similarly, each decoder layer has filters of size 5 and increase the feature maps width and height by 4.

10.1 The model family

The model family we will study in this chapter is a convolutional autoencoder with varying number of layers. The model family consists in a convolutional encoder of L layers, followed by a convolutional decoder of L layers which increases the size of the feature maps and gets back the original image. In the decoder, increasing the size of the feature maps is implemented by using a transposed convolution (Dumoulin & Visin, 2016), which in this case is equivalent to a convolution with zero padding (also called the *full* mode).

We denote the model instances from the proposed model family as the following: $\text{model}(L) = \text{conv}(k) - \text{conv}(k) \ldots \text{conv}(k) - \text{act}(\cdot) - \text{upconv}(k) - \text{upconv}(k) \ldots \text{upconv}(k)$. Each *conv* is a convolutional layer followed by *ReLU* and $\text{act}(\cdot)$ is an activation function that we apply in the bottleneck layer. Here, we use $\text{act}(\cdot) = \text{spatialwta}(\cdot)$. *spatialwta* corresponds to the convolutional spatial Winner-Take-All (WTA) activation used in Makhzani & Frey (2015), also explained in Section 2.2.3. The part from the input layer up to *act* is the encoder and consists in a stack of L convolutional layers. Similarly, the part after *act* is the decoder and has a stack of L convolutional layers with padding to increase the spatial size. k is the size of the filters, we use $k = 5$ for every layer. All the model instances of the model are trained on the MNIST dataset with the reconstruction error objective, using the mean squared error (MSE). After training a model, we generate samples from it exactly like we did in Section 8.5. See Figure 10.1 for an example architecture with $L = 3$.

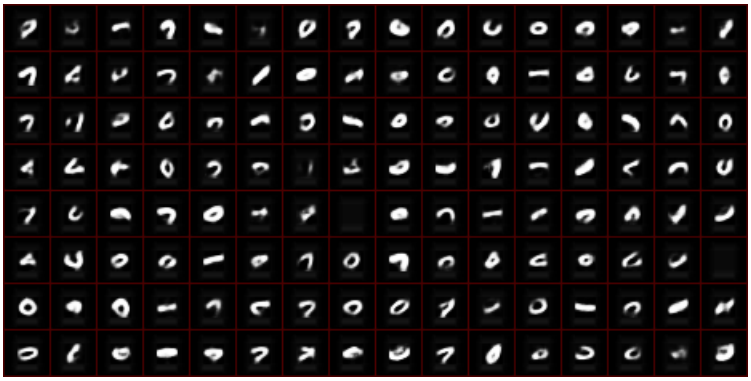


Figure 10.2: Visualization of the total 128 features learned by an instance from the model family with $L = 3$ layers. Each cell correspond to the visualization of one feature. The features correspond to strokes. See the text for more details.

10.2 Visualization of the learned features

Let us take an example of a model from the model family with $L = 3$ layers. After training the model, we can visualize the features learned by the model. Contrary to works on activation maximization (see Section 2.2.5, or Olah et al. (2017)) which provide an indirect visualization of the learned features, the visualization we show here is direct because it is possible to change the values of the activations in the feature maps and see directly their effect on the output without needing any optimization process. We visualize the learned features in the following way. The features we visualize are from the middle layer, which is a volume with the shape $(128, 16, 16)$, as we describe in Figure 10.1. In other words, there are a total of 128 features (corresponding to the depth 128, the number of feature maps) each one represented as a feature map, here of size 16×16. Each feature can either be activated (value greater than zero) or not (value zero) in a given position on the 16×16 grid. Here, we visualize each feature by activating it in the center position, $(8, 8)$, and by zeroing out the other positions and leaving all the other features deactivated (their value is zero in all the locations), then calling the decoder (that is, the rest of the layers, starting from the middle one, see Figure 10.1) to get the reconstructed image. What we show in Figure 10.2 is the reconstructed image we get corresponding to each feature.

We can also see how those features decompose the images into parts. To do that, we use the encoder to compute the features corresponding to an example from MNIST. Then, we show the subset of features that participate into the reconstruction of the example. See Figure 10.3 for an illustration. Each feature is visualized

Figure 10.3: Illustration of how the model decomposes an image into parts. The last image (the digit "4") refers to the reconstructed image. The rest of the images are a visualization of the features that participate into the reconstruction of the digit "4". In this example, only 9 features out of 128 are used to reconstruct the digit "4".

by zeroing out all the remaining features, then calling the decoder, similarly to Figure 10.2.

10.3 Assessment of the ability of models to recognize new patterns

We have seen that the architecture of the model family is capable of decomposing an image into parts (encoder) in the form of features, and recombine the parts into the reconstructed image (decoder). Let us call the images from where the model have been trained as D_{in}, which is MNIST in our case. The images from D_{in}, when fed to the autoencoder, will form a certain set of patterns of activation in the feature space. However, in the context of novelty generation, we are interested into *new* patterns of activation, that are not seen in D_{in}. That is, we would like to recombine the features in a way that does not form a digit. One question we can ask now is, given a trained model, does it allow all the possible patterns in the feature space, or there are certain patterns that are forbidden and others that are preferred? One way to test this would be to see whether the model can recognize patterns unseen in the dataset. Thus, we will estimate the number of patterns that a model "likes" by injecting samples from classes unseen to the model as input x, and see whether that input is a fixed point of the autoencoder. That is, given $x \in D_{\text{out}}$, where D_{out} are images from categories unseen to the model, we would like to compare x and $\text{dec}(\text{enc}(x))$. Suppose $e(x) = \|x - \text{dec}(\text{enc}(x))\|_1$. We propose to use $\text{recRatio}(D_{\text{out}}) = \frac{1}{|D_{\text{out}}|} \sum_{x \in D_{\text{out}}} \mathbb{I}\{e(x) < \theta\}$, where θ is a threshold[1], for estimating the ratio of samples from D_{out} that are "recognized" by the model, which acts as a proxy of the ratio of new patterns that the model can recognize. Note that a good model with respect to $\text{recRatio}(D_{\text{out}})$ does not provide us in any way a mechanism for generating the samples from D_{out}, the score we propose is only a way to check whether the feature space learned by the model allows patterns that

[1]The value of θ itself is not very important as our goal is just to compare different models. We choose a value of $\theta = 50$.

are not seen in D_{in} or forbid them. In fact, the identity function $f(x) = x$ allows us to have a score which is perfect (recRatio$(D_{\text{out}}) = 1.0$) for any D_{out}, without the need of any learning. However, by training the models on D_{in}, and by further visualizing the learned features[2], we can ensure that the model did not just learn the identity function. To summarize, by scoring models using recRatio$(\cdot)$, our aim is to check whether a trained model can reconstruct well images from D_{in}, but also images that are formed by new combinations of the features learned on D_{in}.

First, we will show that it is not trivial that a model can reconstruct very well samples from new classes. To do that, we will first train a denoising fully connected autoencoder with one layer of 1000 units on D_{in}, using the *salt and pepper* (Vincent et al., 2010) noise with a corruption probability of $\frac{1}{2}$. For D_{out}, we choose the HWRT handwritten dataset of mathematical symbols (Thoma, 2015). We choose this dataset because it has a large number of classes (369 classes), and is thus very diverse. Figure 10.4 shows examples from the HWRT dataset. Figure 10.5 shows the original and the reconstructed images.We can notice in Figure 10.5 that some strokes are suppressed or "grayed" by the autoencoder, suggesting that there are some combinations of strokes that it does not allow. To further see whether the autoencoder "likes" those samples, we can check whether they are attractors (see Section 2.2.3). To do that, we feed back the reconstructed image as input to the autoencoder, and reconstruct again. We repeat the same process for 30 iterations, for each image. As we can see in Figure 10.6, all the images end up being transformed to digits, thus the original images are not attractors.

10.4 The ability to recognize new patterns is affected by model capacity

Here, we will show that model capacity affects the ability of a model to recognize new patterns, unseen in D_{in}. Going back to our model family, we will study how constraining capacity of a model in different ways can lead to a consistent decrease in terms of recRatio(D_{out}).

10.4.1 Bottleneck size

Here, we will train the same architecture with $L = 3$, but we vary the number of feature maps in the bottleneck layer (the layer in the middle in Figure 10.1). For

[2]This is a common practice in the autoencoder literature to check whether a model did in fact learn useful features from the data. Another common practice is to test the learned features on a supervised problem, and check the accuracy.

Figure 10.4: Examples from the HWRT dataset (Thoma, 2015). The examples of this dataset are originally represented in a vector format, as a sequence of coordinates (x, y). Thus, we built a rasterized version with the same size than MNIST (28×28).

Figure 10.5: Inputs and reconstructions of samples from the HWRT dataset (Thoma, 2015) obtained using a denoising autoencoder trained on MNIST. Input images are in the first row. The corresponding reconstructed images are in the second row.

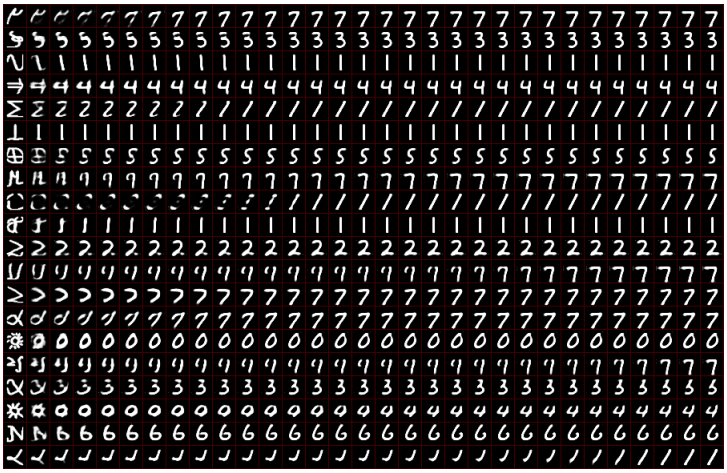


Figure 10.6: Illustration of the reconstruction chain using a denoising autoencoder. We start by an example from the HWRT dataset (Thoma, 2015), reconstruct using the autoencoder, then feed back the reconstruted image to the autoencoder. We repeat the process for 30 iterations. Each row correspond to an example. In the first column we have the original images. The rest of the columns are the reconstructed images in each iteration.

each bottleneck size, we train a model on MNIST, then compute recRatio$(\cdot)$. Figure 10.7 shows the effect of the size of the bottleneck (number of feature maps) on recRatio(D_{in}) and recRatio(D_{out}). We can see that with a bottleneck size from 16 to 64, recRatio(D_{in}) stays constant at 1.0, while recRatio(D_{out}) decreases considerably. When the bottleneck size starts to be extremely small with size 8 and lower, recRatio(D_{in}) starts to decrease as well. The three models with a bottleneck size from 16 to 64 are equivalent with respect to the score recRatio(D_{in}), but very different with respect to the score recRatio(D_{out}).

10.4.2 Sparsity

Here, we will constrain the capacity of the model in a different way. We will use an additional sparsity activation function on the bottleneck (the layer in the middle of Figure 10.1). The sparsity activation function is parametrized by a sparsity rate $0 \leq \rho \leq 1$, and is applied after spatialwta$(\cdot)$. The sparsity activation zeroes out a ratio of ρ feature maps with the smallest activations, leaving a ratio of $(1 - \rho)$ of the feature maps with the biggest activations. We will train the same architecture with $L = 3$, but we will vary ρ. For each value of ρ, we train a model on MNIST, then compute recRatio$(\cdot)$. Figure 10.8 shows the effect of ρ on recRatio(D_{in}) and

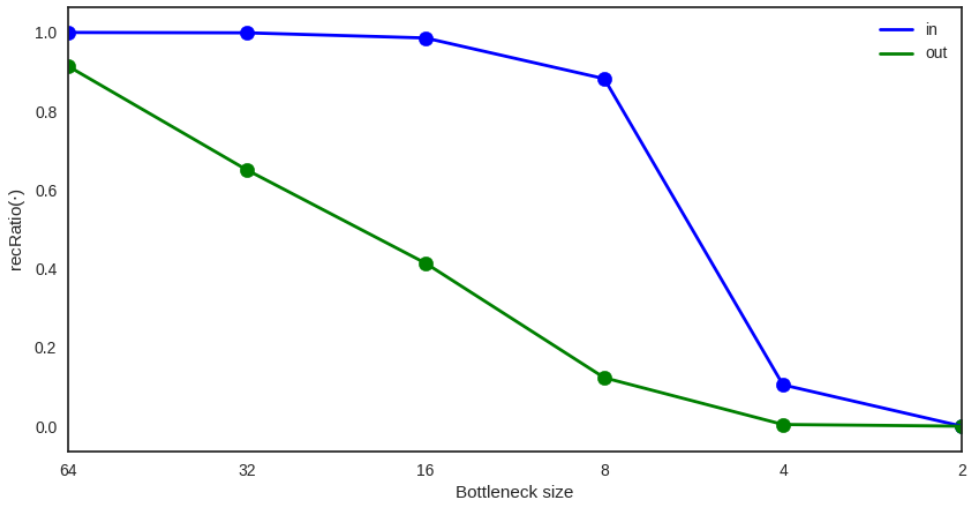


Figure 10.7: Effect of the size of the bottleneck (number of feature maps) on recRatio$(\cdot)$. The blue line refers to to recRatio(D_{in}), which is evaluated on the MNIST test data. The green line refers to to recRatio(D_{out}), which is evaluated on the HWRT (Thoma, 2015) dataset.

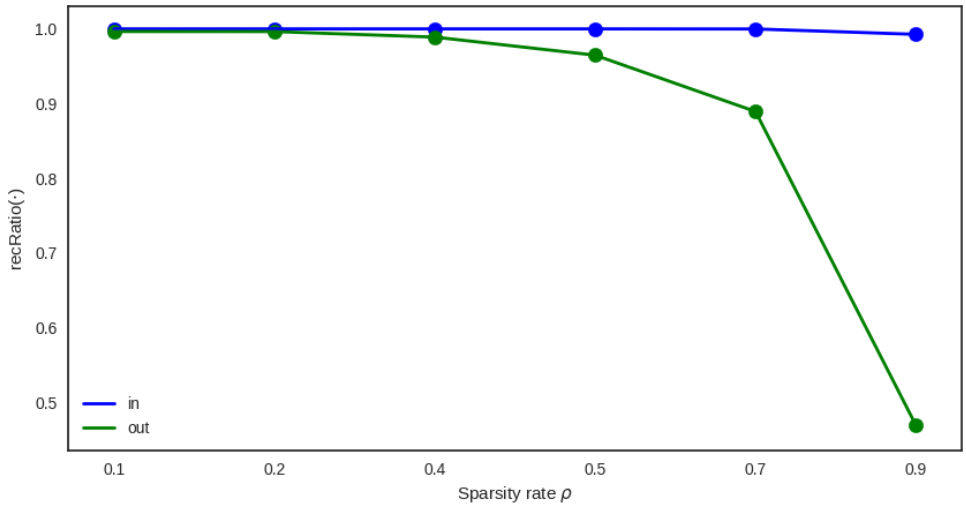


Figure 10.8: Effect of the sparsity rate ρ on recRatio$(\cdot)$. The blue line refers to to recRatio(D_{in}), which is evaluated on the MNIST test data. The green line refers to to recRatio(D_{out}), which is evaluated on the HWRT (Thoma, 2015) dataset.

recRatio(D_{out}). As we can see, as we increase the sparsity rate, recRatio(D_{out}) decreases, while recRatio(D_{in}) stays almost intact. All the models are equivalent with respect to recRatio(D_{in}), but very different with respect to recRatio(D_{out}).

10.4.3 Noise

Here, rather than constraining directly the capacity the model, we are going to constrain the amount of information that is given to the model for reconstructing the images. We implement this by learning a denoising autoencoder, for which the goal is to reconstruct an image from a corruputed version of it. We will use the *salt and pepper* noise (Vincent et al., 2010) with varying probabilities of corruption, $p_{\text{corruption}}$. We will train the same architecture with $L = 3$, but we will vary $p_{\text{corruption}}$. For each value of $p_{\text{corruption}}$, we train a model on MNIST, then compute recRatio$(\cdot)$. Figure 10.9 shows the effect of $p_{\text{corruption}}$ on recRatio(D_{in}) and recRatio(D_{out}). As we can see, by increasing the probability of corruption, both recRatio(D_{in}) and recRatio(D_{out}) decrease, but recRatio(D_{out}) decreases more quickly and has much smaller score than recRatio(D_{in}) .

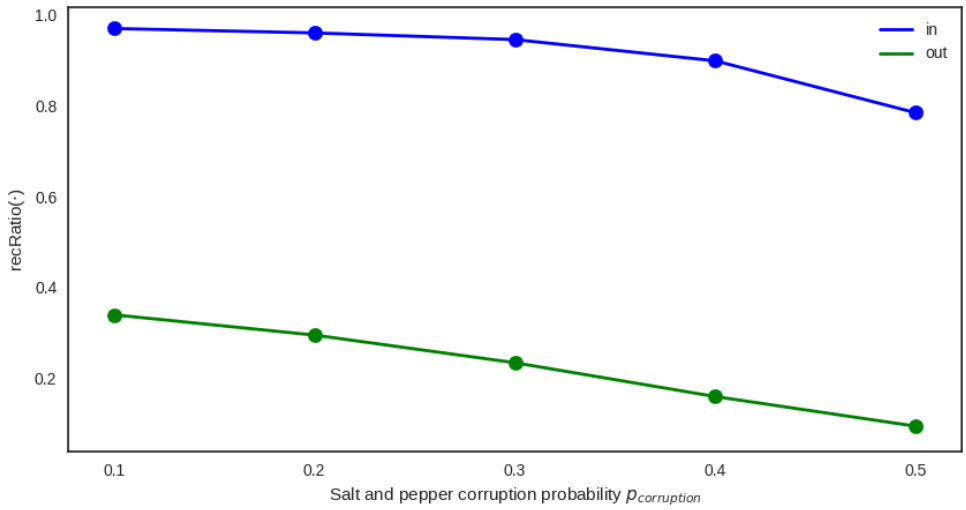


Figure 10.9: Effect of increasing the corruption probability on recRatio$(\cdot)$. The blue line refers to to recRatio(D_{in}), which is evaluated on the MNIST test data. The green line refers to to recRatio(D_{out}), which is evaluated on the HWRT (Thoma, 2015) dataset.

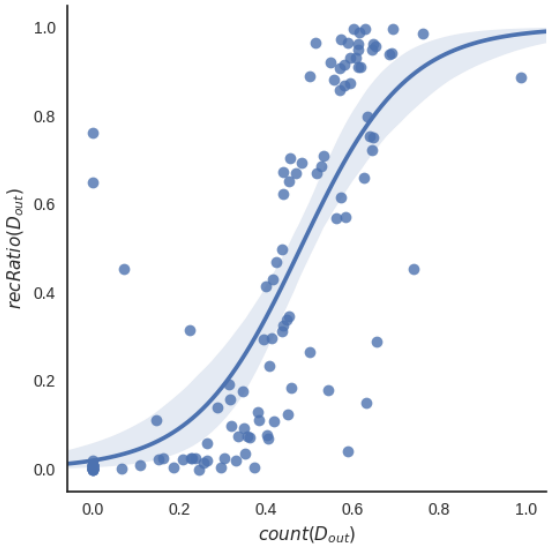


Figure 10.10: Out-of-class count (Chapter 7), a proxy of the ratio of the *generated* samples from unseen classes, against $recRatio(D_{out})$, a proxy of the ratio of samples from unseen classes that the model can recognize. Each point correspond to a model. The blue curve is a logistic regression fit.

10.5 Models that can generate novelty are also good at recognizing new patterns

We have shown previously the capacity of a model affects its ability to recognize new patterns. Here, we show that good out-of-class generators, as we defined them in Chapter 7, are also good at recognizing new patterns, better than in-class generators. To show that, we plot the out-of-class count metric (see Chapter 7), a proxy of the ratio of *generated images* from unseen classes, against $recRatio(D_{\text{out}})$, a proxy of the ratio of patterns from unseen classes that the model can recognize. To show that, we generate images from all the models (a total of 165 models) we trained in Section 10.3. In those models, we varied the bottleneck size, sparsity rate, and the corruption probability. Moreover, we also very the number of layers, from $L = 1 \ldots 6$. Figure 10.10 shows that the more models are generating novelty, the more they can recognize images from unseen classes. Intuitively, the figure suggests that for models to generate novelty, they need to be able to *represent* new patterns not seen in training data, otherwise, if they can only represent patterns that correspond to digits (MNIST), they would not have been able to generate novelty.

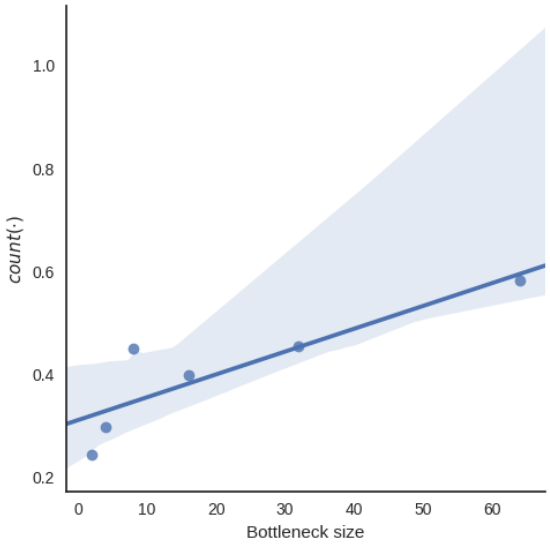


Figure 10.11: Effect of the number of feature maps of the bottleneck on the out-of-class count. The blue curve is a logistic regression fit.

10.6 Restricting the capacity of the models can kill novelty

Here, we show the effect of restricting the capacity on novelty generation. We use the same models that we trained on Section 10.3. In Figure 10.11, we show the effect of the size of the bottleneck on out-of-class count. In Figure 10.12, we show the effect of sparsity on out-of-class count. In Figure 10.13, we show the effect of the corruption on out-of-class count. The three figures suggest that restricting the capacity of the model can be harmful for novelty generation.

To see more concretely the effect of capacity restriction, we show the evolution of the generated images as the capacity is restricted. Figure 10.14 shows the effect of increasing the probability of corruption on the generated images. We can see that as the probability of corruption increases, the model generate more and more digits.

10.7 Conclusion

In Chapter 8, we have demonstrated through hyperparameter search that it is possible to find models that can generate novelty in a setup where we train models on

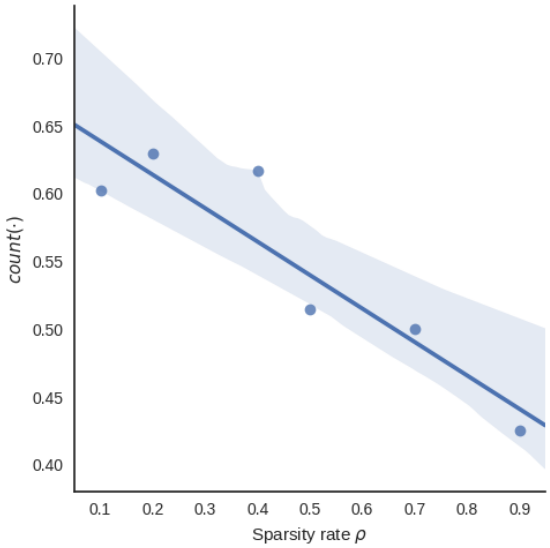


Figure 10.12: Effect of sparsity rate on out-of-class count. The blue curve is a logistic regression fit.

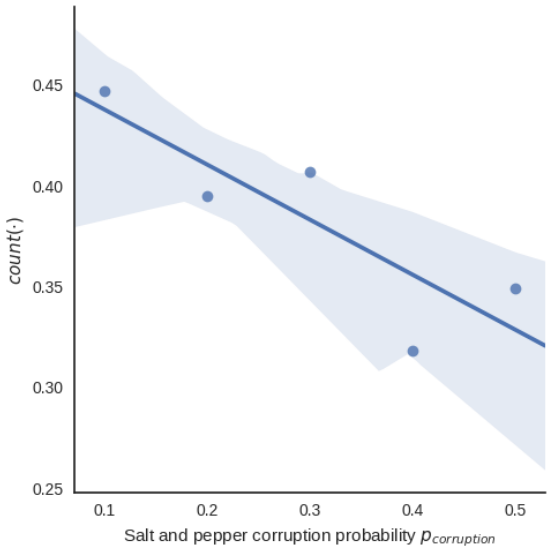


Figure 10.13: Effect of corruption on out-of-class count. The blue curve is a logistic regression fit.

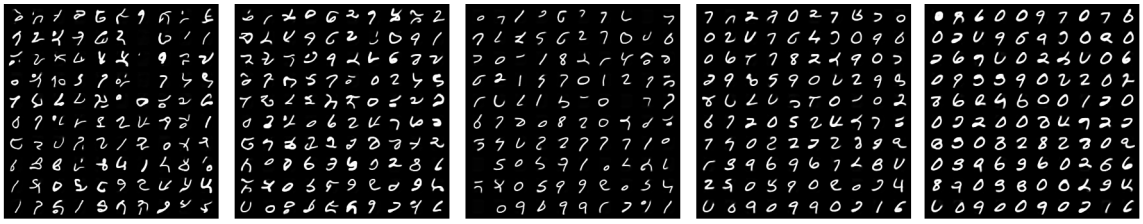


Figure 10.14: Evolution of the generated images as the probability of corruption increases. Each grid of images correspond to 100 generated images from a model trained on MNIST with the corresponding probability of corruption. All the models have the same hyperparameters, except the probability of corruption.

digits and find models that can generate letters, despite having never seen letters. We also found as well models that are good to regenerate known objects only, digits in our case. In Section 4.1, we argued through a series of examples and then through a literature review that compressing the representation can decrease the capacity of the models to generate novelty. This chapter confirms experimentally that it indeed happens in practice through a series of experiments where we used convolutional sparse autoencoders. We find that hyperparameters that control the capacity of the models, such as sparsity and the size of the bottleneck and corruption probability, can reduce the ability of the models to generate novelty, constraining them to generate only known objects. In other words, if the representation space is too much constrained, the model have no other choice than representing only the known objects, due to the reconstruction error pressure (in an autoencoder). As a consequence, the model cannot generate novelty, because it cannot represent objects which do not correspond to known classes.

Chapter 11

Conclusion and perspectives

In this thesis, we proposed to study novelty generation within the frame of machine learning. We have seen that in the machine learning literature, the problem of finding a good representation is seen as crucial, since it can make the learning problem much easier. However, much of the work is about finding good representations for the prediction task. On the other hand nothing prevents us to learn good representations for other tasks such as novelty generation. We argued that while the framework of statistical learning theory is extremely valuable and brought us several advancements in the field, we believe it is not suitable for studying the generation of novelty. We have reviewed two fields that studied the question of novelty generation for decades, namely computational creativity and design theory, and then we raised their current limitations. There are two general issues that we found in the field of computational creativity. First, the generative systems which are used in general either lack a model of knowledge or use a hand-encoded model of knowledge. The second issue is what we called the fitness function barrier, which means that it is the programmer of the system who defines the value function and, as a result, the generated objects reflect more the choices of the designer of the fitness function rather than the machine. We have also reviewed design theory and we have seen that it is a theoretical field which studies the design process, but there are no experimental tools which can be used to study the design process, and there are no discussions about the value function.

There have been several advancements in the generative modeling literature recently. However, most of the works are about learning to re-generate known objects, and the literature is mostly concerned about generating realistic objects rather than studying the foundations or proposing new tasks. This thesis is an attempt to propose foundations to novelty generation as well as a set of contributions on new tasks for generative modeling. To clarify the task of generation, we took a

step back and explained why a good representation can be helpful for generating novelty. The first contribution of the thesis is the clarification of the relationship between knowledge, representation, and value. In particular, we argue that the agent needs to re-represent the data to make generation of novelty easier, and the representation the agent constructs should be guided by previous knowledge as well as a value function, which can be an external or an internal value function (constructed automatically by the agent). We argued that deep learning models can be helpful for novelty generation because they learn a hierarchical representation that constrain mutually the intermediate layers, making it unlikely to generate noise in the output.

In order to study novelty generation, we performed a series of experiments. In Chapter 7, we have observed that a model trained on digits, namely a sparse convolutional autoencoder, could generate new combinations of the strokes learned on the digits, which led to symbols that would not be recognized as digits. We explained this behavior by arguing that the model learned an implicit internal value function, glued into the generation mechanism, which would make valuable the combinations of strokes that could lead to digits as well as the combinations of strokes that could lead to other symbols. Interestingly, the model was trained using a standard objective function used in autoencoders, the reconstruction error. The model we found in Chapter 7 would be a bad model if our goal was to learn a model of the data, because such good model would generate with high probability only examples from the data distribution, in that case digits. In other words, we argue that even if the goal of current objective functions used in machine learning is to model the data, they often do not do so perfectly, which explains why we could find a model that generated new combinations of strokes. Importantly, while such kind of models or generated samples would be ignored or suppressed under the current paradigm of the generative modeling literature because they are considered as *failure modes* or *spurious*, we chose to study them. The second key result of the thesis is the confirmation that current models in the generative literature, even when trained with traditional objective functions, can generate unknown objects.

In Chapter 8, we have proposed evaluation metrics to select models that are good for novelty generation, in order to automatically find models such as the ones we found in Chapter 7. The setup is based on a similar idea of held-out validation in machine learning. We simulate novelty by holding-out entire classes, and finding models that generate samples from the held-out classes while trained on the other (training) classes. The training classes were digits, and the held-out classes were letters. We proposed several metrics that estimate how many samples a model could generate from the held-out classes. We successfully found models that could generate novelty as well as other models that could not. Then, we studied the difference between the models that could generate novelty and the models that could

not. The main finding from the study is that regularization tricks (e.g., sparsity, bottleneck size, corruption) that constrain the capacity of the models also reduce the ability of models to generate novelty. One limitation of our study is that it does not explain why the models can generate novelty, it just shows that by constraining capacity we generate less novelty. In the future we would like to investigate more why a model can generate novelty, and characterize the generated samples and their relationship with the training set. Moreover, in our setup we trained the models with classical machine learning objectives and did hyperparameter optimization to find models that can generate novelty according to our proposed metrics. It would be interesting to have more control on that side, and propose an alternative training criterion that encourages the model to generate novelty naturally. It would be also interesting to have more control on the generation mechanism, as we rely mostly on random sampling.

In Chapter 7 and Chapter 8, our setup was limited to small handwritten characters. While this was a deliberate choice to make the study of novelty generation easier, it would also be very interesting to see whether our work applies to more complicated setups. For instance, we would like to investigate whether the setup can work for larger images with more details. Moreover, we would like to study novelty generation on other artistic domains such as music, poetry, and video game content generation but also on scientific and engineering domains for problems such as drug design and design of materials.

In Chapter 9, we demonstrated the usefulness of novelty generation on a scientific field, namely drug design. In this setup, the goal was to learn a generative model of molecules based on an existing set of drug-like molecules. We have shown that it is possible to find models that can generate molecules better than the existing ones in training set according to a domain specific evaluation criterion. While the setup of this work was simple, it demonstrates that it is beneficial to consider investigating data-driven generative models, in order to do knowledge-driven optimization. This work was a first attempt. To go further, we would like to find a way to train and generate directly molecules that score high according to the evaluation criterion, as currently the generation mechanism is disconnected from the evaluation criterion.

We have proposed a conceptual framework (the KRV framework) to clarify the relationship between knowledge, representation, and value. In the thesis, we did not consider dynamical generation, that is, the state of the agent (knowledge, representation, and value function) did not change over time. It would be interesting as a future avenue to study a setup where the agent can evolve over time, collecting new knowledge, and receiving feedback from the external world and interacting with it. In particular, we know that human design occurs within a community. Even when the design activity is individual (such as painting), there is constant feedback from

the external world such as from critics or the public. It would be interesting to investigate a multi-agent system of designers, where each agent can design objects as well as communicate with other designer agents to give external feedback or gather new knowledge from them.

Appendix A

Synthèse

L'intelligence artificielle (IA) est vue comme technologie qui va profondément changer la société humaine à travers l'automatisation des tâches répétitives, le coût réduit, la possibilité de faire des tâches que les humains excellent (reconnaissance des motifs, contrôle, abstraction) ainsi que des tâches qui sont hors de portée des humains (par exemple la prédiction de météo ou la conception efficace des molécules pour la création de nouveaux matériaux ou de médicaments).

Le domaine de l'intelligence artificielle est vaste et comprends plusieurs sous-domaines comme la planification, l'apprentissage, la perception, le contrôle (du mouvement du corps), la communication (à travers le langage naturel), la satisfaction de contraintes.

Grâce à son succès empirique, l'apprentissage automatique est considéré comme le domaine phare de l'IA. Les plus grands succès de l'apprentissage automatique ont été dans la prédiction. Pour plusieurs problèmes pratiques et dans un paradigme d'ingénierie, les tâches de prédiction sont extrêmement utiles. En effet, le paradigme de prédiction a permis l'accès à des technologies novatrices comme la première génération des voitures autonomes ou les assistants personnels intelligents (par exemple Siri ou Amazon Echo). Dans la recherche, Les méthodes de prédiction sont basées sur un cadre bien compris qui est la théorie de l'apprentissage statistique. L'apprentissage statistique fournit des définitions rigoureuses (de ce qu'est l'apprentissage par exemple) ainsi que des moyens pour comparer les différentes méthodes et mesurer le progrès.

Cependant, les méthodes actuelles d'IA basées sur la prédiction ignorent et échouent à reproduire plusieurs autres traits de l'intelligence humaine comme les émotions, la compréhension intuitive de la physique, la psychologie, la capacité d'apprendre à partir de très peu d'exemples ou de démonstrations (Lake et al., 2017). En choisissant de ne pas (ou peu) considérer ces traits essentiels, l'apprentissage

automatique présente finalement peu d'intérêt pour étudier expérimentalement une IA qui peut aller au delà du simple (mais puissant) paradigme de prédiction. Atteindre une IA qui peut reproduire ces traits semblerait donc improbable si la communauté de l'apprentissage automatique continue à négliger ces problèmes et reste concentrée sur les méthodes de prédictions.

Une de autres caractéristiques de l'intelligence humaine, qui est le cœur de cette thèse, est la capacité des humains à créer de la nouveauté. La capacité des humains de créer la nouveauté se manifeste dans plusieurs domaines comme la conception de nouveaux produits (par exemple dans l'ingénierie), l'art (par exemple les peintures, la musique), les œuvres intellectuelles (par exemple les théories scientifiques, romans). Alors que l'impact et la pertinence de la génération de la nouveauté ont été confirmés et soulignés dans de nombreuses littératures scientifiques (psychologie cognitive, gestion de l'innovation, théorie de conception), l'intelligence artificielle en général et l'apprentissage automatique en particulier sont restés insensibles au sujet. La raison principale de ce manque d'intérêt est probablement dû à l'absence d'un cadre théorique définissant clairement ce qu'est la génération de nouveauté et quelles sont les tâches où la génération de nouveauté avec des méthodes d'apprentissage automatique serait utile. Un tel cadre assurerait la productivité ultérieure de la recherche sur le sujet.

Dans cette thèse, on propose d'étudier la génération de la nouveauté dans le cadre de l'apprentissage automatique. Dans la littérature de l'apprentissage automatique, trouver une bonne représentation des données est crucial parce qu'une bonne représentation peut rendre l'apprentissage beaucoup plus performant. Cependant, la plupart des travaux étudient comment trouver des bonnes représentations pour la tâche de prédiction. La thèse pose la question de comment apprendre une bonne représentation pour la génération de la nouveauté. D'un autre côté, bien que le cadre théorique de l'apprentissage statistique soit extrêmement utile, on argumente que ce cadre théorique n'est pas adapté à la génération de la nouveauté. Dans la thèse on fait une revue de littérature sur deux champs liés à la génération de nouveauté : créativité computationnelle et théorie de conception et relève leurs limitations.

Il y a deux problèmes relevés dans la littérature de la créativité computationnelle. En premier, les systèmes génératifs utilisés dans la créativité computationnelle en général manquent d'un modèle de connaissances ou utilisent un modèle de connaissance développé à la main. Le deuxième problème concerne ce qu'on appelle par "fitness function barrier", qui veut dire que comme c'est le programmeur ou concepteur du système qui fixe la fonction de valeur des objets, le résultat des objets générés reflète plus les choix du concepteur de la fonction de valeur plutôt que ceux de la machine. On fait également une revue de littérature de la théorie de conception qui est un champ théorique qui étudie le processus de conception, mais

il n'y a pas de proposition d'outils expérimentaux qui peuvent être utilisés pour étudier le processus de conception et il y a peu de discussions sur la fonction de valeur.

Récemment, il y a eu plusieurs avancements sur la littérature des modèles génératifs basés sur l'apprentissage automatique. Cependant, la plupart des travaux étudient comment regénérer des objets connus et s'occupe généralement de comment les rendre plus réalistes plutôt que d'étudier les fondations et proposer de nouvelles tâches liées à la génération. Cette thèse est une tentative de poser les fondations de la génération de nouveauté et propose de nouvelles tâches pour les modèles génératifs. Pour clarifier la tâche de génération, on commence par expliquer pourquoi une bonne représentation est utile pour la génération de la nouveauté. La première contribution de la thèse est la clarification entre les concepts de connaissances, représentation et valeur. En particulier, la thèse argumente qu'un agent concepteur doit rereprésenter les données pour faciliter la génération et la représentation que l'agent construit doit être guidée par les connaissances et la fonction de valeur qui peut être interne ou externe. Dans la thèse on argumente que les modèles génératifs profonds peuvent être utiles pour la génération de la nouveauté parce qu'ils apprennent une représentation hiérarchique qui contraint mutuellement les couches intermédiaires, rendant la génération de bruit dans la sortie improbable. Pour étudier la génération de la nouveauté, on fait une série d'expérimentations.

Dans une première expérimentation, un modèle "sparse convolutional autoencoder" est entraîné sur un jeu de données d'images de chiffres manuscrits et on observe que le modèle pouvait générer de nouvelles combinaisons de traits appris sur les chiffres qui ont permis de pouvoir générer des symboles qui ne sont pas reconnaissables en tant que chiffres. Ce comportement peut être expliqué par le fait que le modèle ait implicitement appris une fonction de valeur interne couplée au mécanisme de génération qui acceptait les combinaisons de traits qui pouvaient former des chiffres mais aussi d'autres symboles. Le modèle a été entraîné en utilisant une fonction objective standard qui est la minimisation d'erreur de reconstruction. Le modèle trouvé serait considéré mauvais si le but était d'apprendre strictement un modèle du domaine de chiffres parce que idéalement on voudrait trouver un modèle qui va générer avec une probabilité élevée les exemples du domaine, ici les chiffres, et avec une très petite probabilité tout le reste. Autrement dit, même si le but actuel des fonctions objectives utilisées en apprentissage automatique est d'apprendre un modèle pour un domaine donné (par exemple le domaine de chiffres), ce but n'est pas atteint parfaitement, ce qui explique l'existence d'un modèle qui génère des nouvelles combinaisons de traits. Ces modèles là ont tendance à être ignorés ou supprimés sous le paradigme actuel des modèles génératifs parce qu'ils sont considérés comme des anomalies, alors que le but de la thèse est précisément de les

étudier. Le deuxième résultat clé de la thèse est donc la conformation que les modèles actuels de la littérature peuvent générer des objets non reconnaissables même s'ils sont entraînés avec des fonctions objectives traditionnelles.

Dans une deuxième expérimentation, on propose des métriques d'évaluation pour détecter et sélectionner les modèles qui sont bons pour la génération de nouveauté pour pouvoir automatiquement trouver des modèles comme celui qui a été trouvé dans la première expérimentation. L'idée est similaire à la méthode de validation des modèles en apprentissage automatique où les données totales sont divisées en deux sous-ensembles, un ensemble d'entraînement et un ensemble de test. En apprentissage automatique, le modèle est généralement entraîné sur l'ensemble d'entraînement et la performance du modèle sur les données non vues/futures est simulée sur l'ensemble du test. L'idée ici est similaire, sauf qu'elle est faite au niveau des catégories : l'ensemble des catégories disponibles est divisé en deux, le modèle génératif est entraîné sur des catégories d'entraînement et l'objectif est de trouver des modèles qui ont la possibilité de générer des exemples de catégories non vues qui sont simulées par l'ensemble de catégories de test. Les catégories d'entraînement étaient des chiffres et les catégories de test étaient les lettres. On a proposé des métriques pour estimer le nombre d'images que le modèle pouvait générer à partir des catégories non vues par le modèle (ici, des lettres). L'expérimentation démontre qu'il est tout à fait possible de trouver des modèles qui génèrent des catégories non vues (les lettres) ainsi que d'autres modèles qui génèrent seulement les catégories vues (les chiffres).

Nous démontrons ensuite l'utilité de la génération de nouveauté dans un domaine scientifique, où l'objectif est de générer des molécules pour la conception de médicaments. Dans la littérature classique de la conception de médicaments, la plupart des techniques de génération de structures moléculaires reposent sur une bibliothèque de fragments, qui sont des sous-graphes du graphe moléculaire. Les fragments sont ensuite combinés de pour former de nouvelles molécules, en utilisant un ensemble de règles, créés par des spécialistes, qui peuvent néanmoins aboutir à un espace de recherche limité ou biaisé. Dans cette expérimentation, nous étudions l'utilisation de modèles d'apprentissage profonds pour générer de nouvelles molécules basées sur un ensemble de molécules semblables à des médicaments existants. Dans ce cas là, les règles sont apprises automatiquement à partir des données plutôt que conçues par des humains. Nous utilisons une fonction d'évaluation spécifique au domaine pour trouver des modèles qui génèrent des molécules qui peuvent potentiellement aider à la création de médicaments et nous trouvons des modèles qui peuvent générer des molécules qui sont meilleures que celles existantes dans l'ensemble d'apprentissage selon la fonction d'évaluation.

La deuxième expérimentation démontre qu'il est possible de trouver des modèles qui génèrent des catégories non vues (les lettres) ainsi que d'autres modèles

qui génèrent seulement les catégories vues (les chiffres). Nous proposons ensuite d'étudier la différence entre ces deux types de modèles. Le principal et dernier résultat de cette thèse est que contraindre la capacité des modèles (à travers différents mécanismes comme la sparsité, bruitage des données ou la contrainte du nombre de neurones) peuvent réduire la capacité des modèles à générer de la nouveauté.

Bibliography

Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, Li Deng, Gerald Penn, and Dong Yu. Convolutional neural networks for speech recognition. *IEEE/ACM Transactions on audio, speech, and language processing*, 22(10): 1533–1545, 2014.

Guillaume Alain and Yoshua Bengio. What regularized auto-encoders learn from the data-generating distribution. *Journal of Machine Learning Research*, 15 (1):3563–3593, 2014.

Dzmitry Bahdanau and Herbert Jaeger. Smart decisions by small adjust-ments: Iterating denoising autoencoders. Technical report, Technical Report 32, Jacobs University, School of Engineering and Science, 2014.

Pierre Baldi and Kurt Hornik. Neural networks and principal component analysis: Learning from examples without local minima. *Neural networks*, 2(1):53–58, 1989.

Virginie Beaucousin, Mathieu Cassotti, Grégory Simon, Arlette Pineau, Milena Kostova, Olivier Houdé, and Nicolas Poirel. Erp evidence of a meaningfulness impact on visual global/local processing: when meaning captures attention. *Neuropsychologia*, 49(5):1258–1266, 2011.

Normand J Beaudry and Renato Renner. An intuitive proof of the data processing inequality. *arXiv preprint arXiv:1107.0740*, 2011.

Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957. URL http://books.google.com/books?id=fyVtp3EMxasC&pg=PR5&dq=dynamic+programming+richard+e+bellman&client=firefox-a#v=onepage&q=dynamic%20programming%20richard%20e%20bellman&f=false.

Yoshua Bengio and Yves Grandvalet. No unbiased estimator of the variance of k-fold cross-validation. *Journal of machine learning research*, 5(Sep):1089–1105, 2004.

Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb): 1137–1155, 2003.

Yoshua Bengio, Pascal Lamblin, Dan Popovici, and Hugo Larochelle. Greedy layer-wise training of deep networks. In B. Schölkopf, J. C. Platt, and T. Hoffman (eds.), *Advances in Neural Information Processing Systems 19*, pp. 153–160. MIT Press, 2007. URL http://papers.nips.cc/paper/3048-greedy-layer-wise-training-of-deep-networks.pdf.

Yoshua Bengio, Aaron C Courville, and Pascal Vincent. Unsupervised feature learning and deep learning: A review and new perspectives. *CoRR, abs/1206.5538*, 1, 2012.

Yoshua Bengio, Grégoire Mesnil, Yann Dauphin, and Salah Rifai. Better mixing via deep representations. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 552–560, 2013a.

Yoshua Bengio, Li Yao, Guillaume Alain, and Pascal Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pp. 899–907, 2013b.

James Bergstra, Norman Casagrande, Dumitru Erhan, Douglas Eck, and Balázs Kégl. Aggregate features and adaboost for music classification. *Machine learning*, 65(2-3):473–484, 2006.

George David Birkhoff. *Aesthetic measure*, volume 9. Harvard University Press Cambridge, 1933.

Margaret A Boden and Ernest A Edmonds. What is generative art? *Digital Creativity*, 20(1-2):21–46, 2009.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

André Breton. Marcel duchamp : The bride stripped bare by her own bachelors. *This Quarter « Surrealist Number »*, 5(1), 1932.

Olivier Breuleux, Yoshua Bengio, and Pascal Vincent. Unlearning for better mixing. *Universite de Montreal/DIRO*, 2009.

Richard Buchanan. Wicked problems in design thinking. *Design issues*, 8(2):5–21, 1992.

Dave Burraston and Ernest Edmonds. Cellular automata in generative electronic music and sonic art: a historical and technical review. *Digital Creativity*, 16 (3):165–185, 2005.

Alex J Champandard. Semantic style transfer and turning two-bit doodles into fine artworks. *arXiv preprint arXiv:1603.01768*, 2016.

Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2172–2180, 2016.

Mehdi Cherti, Balázs Kégl, and Akın Kazakçı. Out-of-class novelty generation: an experimental foundation. 2016.

Mehdi Cherti, Balázs Kégl, and Akın Kazakçı. De novo drug design with deep generative models: an empirical study. In *ICLR workshop*, 2017.

Kyunghyun Cho. Boltzmann machines and denoising autoencoders for image denoising. *arXiv preprint arXiv:1301.3468*, 2013.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

François Chollet et al. Keras, 2015.

Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

Kai Lai Chung. *Markov chains*. Springer, 1967.

Jeff Clune and Hod Lipson. Evolving three-dimensional objects with a generative encoding inspired by developmental biology. In *Proceedings of the European Conference on Artificial Life*, pp. 144–148, 2011.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *arXiv preprint arXiv:1702.05373*, 2017.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

John Conway. The game of life. *Scientific American*, 223(4):4, 1970.

Matthew Cook. Universality in elementary cellular automata. *Complex systems*, 15(1):1–40, 2004.

Michael Cook and Simon Colton. Generating code for expressing simple preferences: Moving on from hardcoding and randomness. In *ICCC*, pp. 8–16, 2015.

George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

Ryan Dahl, Mohammad Norouzi, and Jonathon Shlens. Pixel recursive super resolution. *arXiv preprint arXiv:1702.00783*, 2017.

Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z Wang. Studying aesthetics in photographic images using a computational approach. In *European Conference on Computer Vision*, pp. 288–301. Springer, 2006.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. IEEE, 2009.

Emily L Denton, Soumith Chintala, Rob Fergus, et al. Deep generative image models using a laplacian pyramid of adversarial networks. In *Advances in Neural Information Processing Systems*, pp. 1486–1494, 2015.

Luc Devroye. Sample-based non-uniform random variate generation. In *Proceedings of the 18th conference on Winter simulation*, pp. 260–265. ACM, 1986.

Fabian Dey and Amedeo Caflisch. Fragment-based de novo ligand design by multiobjective evolutionary optimization. *Journal of chemical information and modeling*, 48(3):679–690, 2008.

Jennifer Diedrich, Mathias Benedek, Emanuel Jauk, and Aljoscha C Neubauer. Are creative ideas novel and useful? *Psychology of Aesthetics, Creativity, and the Arts*, 9(1):35, 2015.

Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *arXiv preprint arXiv:1501.00092*, 2014.

Alexey Dosovitskiy and Thomas Brox. Generating images with perceptual similarity metrics based on deep networks. In *Advances in Neural Information Processing Systems*, pp. 658–666, 2016.

Alexey Dosovitskiy, Jost Springenberg, Maxim Tatarchenko, and Thomas Brox. Learning to generate chairs, tables and cars with convolutional networks. 2016.

Dominique Douguet, Hélène Munier-Lehmann, Gilles Labesse, and Sylvie Pochet. Lea3d: a computer-aided ligand design for structure-based drug design. *Journal of medicinal chemistry*, 48(7):2457–2468, 2005.

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning. *arXiv preprint arXiv:1603.07285*, 2016.

Sean R Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3): 361–365, 1996.

Ernest Edmonds. Independence of rose's axioms for m-valued implication. *The Journal of Symbolic Logic*, 34(02):283–284, 1969.

Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of cheminformatics*, 1(1):8, 2009.

Mary Farbood and Bernd Schöner. Analysis and synthesis of palestrina-style counterpoint using markov chains. In *ICMC*, 2001.

Gilles Fauconnier and Mark Turner. Blending as a central process of grammar. *Conceptual structure, discourse, and language*, 113:130, 1996.

Uli Fechner and Gisbert Schneider. Flux (1): a virtual synthesis scheme for fragment-based de novo design. *Journal of chemical information and modeling*, 46(2):699–707, 2006.

William T Freeman, Thouis R Jones, and Egon C Pasztor. Example-based super-resolution. *IEEE Computer graphics and Applications*, 22(2):56–65, 2002.

P Galanter. Generative art after computers,[in:] generative art–proceedings ga2012 xv generative art conference, red. soddu c, 2012a.

Philip Galanter. Computational aesthetic evaluation: Past and future. In *Computers and Creativity*, pp. 255–293. Springer, 2012b.

L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. Aug 2015a. URL http://arxiv.org/abs/1508.06576.

Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015b.

Jason Gauci and Kenneth O Stanley. Autonomous evolution of topographic regularities in artificial neural networks. *Neural computation*, 22(7):1860–1898, 2010.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Murray Gell-Mann. What is complexity? In *Complexity and industrial clusters*, pp. 13–24. Springer, 2002.

Zoubin Ghahramani. Unsupervised learning. In *Advanced lectures on machine learning*, pp. 72–112. Springer, 2004.

Valerie Gillet, A Peter Johnson, Pauline Mata, Sandor Sike, and Philip Williams. Sprout: a program for structure generation. *Journal of computer-aided molecular design*, 7(2):127–153, 1993.

Rafael Gómez-Bombarelli, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, David Duvenaud, Dougal Maclaurin, Martin A Blood-Forsythe, Hyun Sik Chae, Markus Einzinger, Dong-Gwang Ha, Tony Wu, et al. Design of efficient molecular organic light-emitting diodes by a high-throughput virtual screening and experimental approach. *Nature Materials*, 15(10):1120–1127, 2016a.

Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D Hirzel, Ryan P Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *arXiv preprint arXiv:1610.02415*, 2016b.

Ian Goodfellow. Nips 2016 tutorial: Generative adversarial networks. *arXiv preprint arXiv:1701.00160*, 2016.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

Gary R Greenfield. Co-evolutionary methods in evolutionary art. *The Art of Artificial Evolution*, pp. 357–380, 2008.

Karol Gregor, Ivo Danihelka, Andriy Mnih, Charles Blundell, and Daan Wierstra. Deep autoregressive networks. *arXiv preprint arXiv:1310.8499*, 2013.

Arthur Gretton, Karsten M Borgwardt, Malte J Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(Mar):723–773, 2012.

Peter Grunwald and Paul Vitányi. Shannon information and kolmogorov complexity. *arXiv preprint cs/0410002*, 2004.

Christian Guckelsberger, Christophe Salge, and Simon Colton. Addressing the "why?" in computational creativity: A non-anthropocentric, minimal model of intentional creative agency. 2017.

Ishaan Gulrajani, Kundan Kumar, Faruk Ahmed, Adrien Ali Taiga, Francesco Visin, David Vazquez, and Aaron Courville. Pixelvae: A latent variable model for natural images. *arXiv preprint arXiv:1611.05013*, 2016.

Markus Hartenfeller and Gisbert Schneider. Enabling future drug discovery by de novo design. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, 1(5):742–759, 2011.

Markus Hartenfeller, Heiko Zettl, Miriam Walter, Matthias Rupp, Felix Reisen, Ewgenij Proschak, Sascha Weggen, Holger Stark, and Gisbert Schneider. Dogs: reaction-driven de novo design of bioactive compounds. *PLoS Comput Biol*, 8(2):e1002380, 2012.

Hatchuel and Weil. A new approach of innovative design: an introduction to ck theory. In *DS 31: Proceedings of ICED 03, the 14th International Conference on Engineering Design, Stockholm*, 2003.

Hatchuel and Weil. Design as forcing: deepening the foundations of ck theory. *Guidelines for a Decision Support Method Adapted to NPD Processes*, 2007.

Armand Hatchuel and Benoit Weil. Ck design theory: an advanced formulation. *Research in engineering design*, 19(4):181–192, 2009.

Armand Hatchuel, Benoit Weil, and Pascal Le Masson. Towards an ontology of design: lessons from c–k design theory and forcing. *Research in engineering design*, 24(2):147–163, 2013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034, 2015.

Geoffrey E Hinton. Distributed representations. 1984.

Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

SR Holtzman. Using generative grammars for music composition. *Computer Music Journal*, 5(1):51–64, 1981.

Proceedings of the International Conference on Computational Creativity, 2016. ICCC.

John J Irwin, Teague Sterling, Michael M Mysinger, Erin S Bolstad, and Ryan G Coleman. Zinc: a free tool to discover chemistry for biology. *Journal of chemical information and modeling*, 52(7):1757–1768, 2012.

Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*, 2016.

Kyle E Jennings. Developing creativity: Artificial barriers in artificial intelligence. *Minds and Machines*, 20(4):489–501, 2010.

Yanghua Jin, Jiakai Zhang, Minjun Li, Yingtao Tian, Huachun Zhu, and Zhihao Fang. Towards the automatic anime characters creation with generative adversarial networks. *arXiv preprint arXiv:1708.05509*, 2017.

Philip N Johnson-Laird. How jazz musicians improvise. *Music Perception: An Interdisciplinary Journal*, 19(3):415–442, 2002.

Donald R Jones, Matthias Schonlau, and William J Welch. Efficient global optimization of expensive black-box functions. *Journal of Global optimization*, 13(4):455–492, 1998.

Nal Kalchbrenner, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu. Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*, 2016.

Hanna Kamyshanska and Roland Memisevic. On autoencoder scoring. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pp. 720–728, 2013.

Hanna Kamyshanska and Roland Memisevic. The potential energy of an autoencoder. *IEEE transactions on pattern analysis and machine intelligence*, 37(6): 1261–1273, 2015.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*, 2015.

Akın Kazakçı. Conceptive artificial intelligence: Insights from design theory. In *International Design Conference DESIGN2014*, pp. 1–16, 2014.

Akın Kazakçı, Mehdi Cherti, and Balázs Kégl. Digits that are not: Generating new types through deep neural nets. In *Proceedings of the Seventh International Conference on Computational Creativity*, 2016.

Akın O Kazakçı. On the imaginative constructivist nature of design: a theoretical approach. *Research in Engineering Design*, 24(2):127–145, 2013.

Akın Osman Kazakçı. *La théorie CKE comme fondement théorique pour les assistants de conception: DesigNAR, un assistant de synthèse de concept basé sur la théorie CKE*. PhD thesis, Université Paris-Dauphine, 2007.

Kazakci, Akin, Hatchuel, Armand, Le Masson, Pascal, Weil, Benoit, et al. Simulation of design reasoning based on ck theory: a model and an example application. In *DS 60: Proceedings of DESIGN 2010, the 11th International Design Conference, Dubrovnik, Croatia*, 2010.

Balázs Kégl. Intrinsic dimension estimation using packing numbers. In *Advances in neural information processing systems*, pp. 697–704, 2003.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Advances in Neural Information Processing Systems*, pp. 3581–3589, 2014.

Mark A Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE journal*, 37(2):233–243, 1991.

Robert J Krawczyk. Architectural interpretation of cellular automata. In *The 5th International Conference on Generative Art*, pp. 7–1, 2002.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Charles W Krueger. Software reuse. *ACM Computing Surveys (CSUR)*, 24(2): 131–183, 1992.

Peter S Kutchukian and Eugene I Shakhnovich. De novo design: balancing novelty and confined chemical space. *Expert opinion on drug discovery*, 5(8):789–812, 2010.

Brenden M Lake, Ruslan Salakhutdinov, and Joshua B Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350 (6266):1332–1338, 2015.

Brenden M Lake, Tomer D Ullman, Joshua B Tenenbaum, and Samuel J Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40, 2017.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. In *Advances In Neural Information Processing Systems*, pp. 4601–4609, 2016.

Greg Landrum. Rdkit: Open-source cheminformatics. *Online). http://www. rdkit. org. Accessed*, 3(04):2012, 2006.

Hugo Larochelle and Iain Murray. The neural autoregressive distribution estimator. AISTATS, 2011.

Yann Lecun and Corinna Cortes.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 05 2015. URL http://dx.doi.org/10.1038/nature14539.

Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, and Wenzhe Shi. Photo-realistic single image super-resolution using a generative adversarial network. *arXiv preprint arXiv:1609.04802*, 2016.

Honglak Lee, Alexis Battle, Rajat Raina, and Andrew Y Ng. Efficient sparse coding algorithms. In *Advances in neural information processing systems*, pp. 801–808, 2007.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pp. 609–616, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553453. URL http://doi.acm.org/10.1145/1553374.1553453.

Joel Lehman and Kenneth O Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary computation*, 19(2):189–223, 2011a.

Joel Lehman and Kenneth O Stanley. Novelty search and the problem with objectives. In *Genetic Programming Theory and Practice IX*, pp. 37–56. Springer, 2011b.

Yan Li, Yuan Zhao, Zhihai Liu, and Renxiao Wang. Automatic tailoring and transplanting: a practical method that makes virtual screening more useful, 2011.

Yan Li, Zhixiong Zhao, Zhihai Liu, Minyi Su, and Renxiao Wang. Autot&t v. 2: An efficient and versatile tool for lead structure generation and optimization. *Journal of chemical information and modeling*, 56(2):435–453, 2016.

Yanghao Li, Naiyan Wang, Jiaying Liu, and Xiaodi Hou. Demystifying neural style transfer. *arXiv preprint arXiv:1701.01036*, 2017.

Antonios Liapis, Héctor Perez Martínez, Julian Togelius, and Georgios N Yannakakis. Transforming exploratory creativity with delenox,. In *ICCC*, pp. 56–63, 2013.

Stefan Lienhard, Cheryl Lau, Pascal Müller, Peter Wonka, and Mark Pauly. Design transformations for rule-based procedural modeling. In *Computer Graphics Forum*, volume 36, pp. 39–48. Wiley Online Library, 2017.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

Róisın Loughran and Michael O'Neill. Application domains considered in computational creativity. 2017.

Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

Penousal Machado and Hugo Amaro. Fitness functions for ant colony paintings. In *ICCC*, pp. 32–39, 2013.

Penousal Machado and Amílcar Cardoso. All the truth about nevar. *Applied Intelligence*, 16(2):101–118, 2002.

Penousal Machado, Juan Romero, María Luisa Santos, Amílcar Cardoso, and Bill Manaris. Adaptive critics for evolutionary artists. In *Workshops on Applications of Evolutionary Computation*, pp. 437–446. Springer, 2004.

Penousal Machado, Juan Romero, and Bill Manaris. Experiments in computational aesthetics. In *The art of artificial evolution*, pp. 381–415. Springer, 2008.

Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. Online dictionary learning for sparse coding. In *Proceedings of the 26th annual international conference on machine learning*, pp. 689–696. ACM, 2009.

Alireza Makhzani and Brendan Frey. k-sparse autoencoders. *arXiv preprint arXiv:1312.5663*, 2013.

Alireza Makhzani and Brendan J Frey. Winner-take-all autoencoders. In *Advances in Neural Information Processing Systems*, pp. 2773–2781, 2015.

Bill Manaris, Dallas Vaughan, Christopher Wagner, Juan Romero, and Robert B Davis. Evolutionary music and the zipf-mandelbrot law: Developing fitness functions for pleasant music. In *EvoWorkshops*, volume 2611, pp. 522–534. Springer, 2003.

Bill Manaris, Juan Romero, Penousal Machado, Dwight Krehbiel, Timothy Hirzel, Walter Pharr, and Robert B Davis. Zipf's law, music classification, and aesthetics. *Computer Music Journal*, 29(1):55–69, 2005.

Brian B Masek, David S Baker, Roman J Dorfman, Karen DuBrucq, Victoria C Francis, Stephan Nagy, Bree L Richey, and Farhad Soltanshahi. Multistep reaction based de novo drug design: Generating synthetically feasible design ideas. *Journal of chemical information and modeling*, 56(4):605–620, 2016.

Jon McCormack. Grammar based music composition. *Complex systems*, 96:321–336, 1996.

Jon McCormack. *Aesthetics, art, evolution*. Springer, 2013.

Jon McCormack and Oliver Bown. Life's what you make: Niche construction and evolutionary art. *Applications of Evolutionary Computing*, pp. 528–537, 2009.

Jon McCormack, Oliver Bown, Alan Dorin, Jonathan McCabe, Gordon Monro, and Mitchell Whitelaw. Ten questions concerning generative computer art. *Leonardo*, 47(2):135–141, 2014.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pp. 3111–3119, 2013.

Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*, 2014.

J Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pp. 400–404. Springer, 1975.

Abraham Moles. Information theory and esthetic perception. 1968.

Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Inceptionism: Going deeper into neural networks. *Google Research Blog. Retrieved June*, 20, 2015.

J-B Mouret and Stéphane Doncieux. Encouraging behavioral diversity in evolutionary robotics: An empirical study. *Evolutionary computation*, 20(1):91–133, 2012.

Georg Nees. *Generative Computergraphik*. Siemens AG, 1969.

Craig G Nevill-Manning, Ian H Witten, and David L Maulsby. Compression by induction of hierarchical grammars. In *Data Compression Conference, 1994. DCC'94. Proceedings*, pp. 244–253. IEEE, 1994.

Andrew Ng. Sparse autoencoder. *CS294A Lecture notes*, 72(2011):1–19, 2011.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 427–436, 2015a.

Anh Nguyen, Alexey Dosovitskiy, Jason Yosinski, Thomas Brox, and Jeff Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. In *Advances in Neural Information Processing Systems*, pp. 3387–3395, 2016a.

Anh Nguyen, Jason Yosinski, Yoshua Bengio, Alexey Dosovitskiy, and Jeff Clune. Plug & play generative networks: Conditional iterative generation of images in latent space. *arXiv preprint arXiv:1612.00005*, 2016b.

Anh Nguyen, Jason Yosinski, and Jeff Clune. Multifaceted feature visualization: Uncovering the different types of features learned by each neuron in deep neural networks. *arXiv preprint arXiv:1602.03616*, 2016c.

Anh Mai Nguyen, Jason Yosinski, and Jeff Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference*, pp. 959–966. ACM, 2015b.

Michael A Nielsen. Neural networks and deep learning, 2015.

Augustus Odena, Christopher Olah, and Jonathon Shlens. Conditional image synthesis with auxiliary classifier gans. *arXiv preprint arXiv:1610.09585*, 2016.

Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. doi: 10.23915/distill.00007. https://distill.pub/2017/feature-visualization.

Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

François Pachet. Musical virtuosity and creativity. In *Computers and creativity*, pp. 115–146. Springer, 2012.

François Pachet and Pierre Roy. Markov constraints: steerable generation of markov sequences. *Constraints*, 16(2):148–172, 2011.

Jussi Parikka. Leonardo book review: The art of artificial evolution: A handbook on evolutionary art and music. 2008.

Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 301–308. ACM, 2001.

Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. Understanding the exploding gradient problem. *CoRR, abs/1211.5063*, 2012.

Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pp. 1532–1543, 2014.

Albert C Pierce, Govinda Rao, and Guy W Bemis. Breed: Generating novel inhibitors through hybridization of known ligands. application to cdk2, p38, and hiv protease. *Journal of medicinal chemistry*, 47(11):2768–2775, 2004.

Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The algorithmic beauty of plants*. Springer Science & Business Media, 2012.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, pp. 3532–3540, 2015.

Scott E Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw. In *Advances In Neural Information Processing Systems*, pp. 217–225, 2016.

Yoram Reich. A critical review of general design theory. *Research in Engineering Design*, 7(1):1–18, 1995.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pp. 91–99, 2015.

Danilo Jimenez Rezende, Shakir Mohamed, Ivo Danihelka, Karol Gregor, and Daan Wierstra. One-shot generalization in deep generative models. In *ICML*, 2016.

Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot, and Yoshua Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pp. 833–840, 2011.

Horst WJ Rittel and Melvin M Webber. Dilemmas in a general theory of planning. *Policy sciences*, 4(2):155–169, 1973.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *NATURE*, 323:9, 1986.

Mark A Runco and Garrett J Jaeger. The standard definition of creativity. *Creativity Research Journal*, 24(1):92–96, 2012.

Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *International Conference on Artificial Intelligence and Statistics*, pp. 448–455, 2009.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. *arXiv preprint arXiv:1606.03498*, 2016.

Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pp. 92–101, 2010.

Jürgen Schmidhuber. *Driven by Compression Progress: A Simple Principle Explains Essential Aspects of Subjective Beauty, Novelty, Surprise, Interestingness, Attention, Curiosity, Creativity, Art, Science, Music, Jokes*, pp. 48–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-02565-5. doi: 10.1007/978-3-642-02565-5_4. URL http://dx.doi.org/10.1007/978-3-642-02565-5_4.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3): 230–247, 2010.

Gisbert Schneider. De novo design–hop (p) ing against hope. *Drug Discovery Today: Technologies*, 10(4):e453–e460, 2013.

Petra Schneider and Gisbert Schneider. De novo design at the edge of chaos: Miniperspective. *Journal of medicinal chemistry*, 59(9):4077–4086, 2016.

Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. Picbreeder: evolving pictures collaboratively online. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 1759–1768. ACM, 2008.

Marwin HS Segler, Thierry Kogej, Christian Tyrchan, and Mark P Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *arXiv preprint arXiv:1701.01329*, 2017.

Claude E Shannon. A mathematical theory of communication, part i, part ii. *Bell Syst. Tech. J.*, 27:623–656, 1948.

Claude E Shannon. Coding theorems for a discrete source with a fidelity criterion. *IRE Nat. Conv. Rec*, 4(142-163):1, 1959.

Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 806–813, 2014.

Herbert A Simon. The sciences of the artificial. *Cambridge, MA*, 1969.

Karl Sims. *Artificial evolution for computer graphics*, volume 25. ACM, 1991.

Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 15–22. ACM, 1994.

Kenneth O Stanley. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, 8(2): 131–162, 2007.

Kenneth O Stanley and Risto Miikkulainen. Efficient evolution of neural network topologies. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pp. 1757–1762. IEEE, 2002.

Teague Sterling and John J Irwin. Zinc 15-ligand discovery for everyone. 2015.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1701–1708, 2014.

Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capabilities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9): 1275–1296, 2001.

Hideaki Takeda, Paul Veerkamp, and Hiroyuki Yoshikawa. Modeling design process. *AI magazine*, 11(4):37, 1990.

RP Taylor. Chaos, fractals, nature: a new look at jackson pollock. *Eugine: Fractal Research*, 170:229, 2006.

Stephen L Thaler. The emerging intelligence and its critical look at us. *Journal of Near-Death Studies*, 17(1):21–29, 1998.

Lucas Theis and Matthias Bethge. Generative image modeling using spatial LSTMs. In *Advances in Neural Information Processing Systems*, pp. 1918–1926, 2015.

Lucas Theis, Aäron van den Oord, and Matthias Bethge. A note on the evaluation of generative models. *arXiv preprint arXiv:1511.01844*, 2015.

Martin Thoma. On-line recognition of handwritten mathematical symbols. *arXiv preprint arXiv:1511.09030*, 2015.

Peter M Todd. A connectionist approach to algorithmic composition. *Computer Music Journal*, 13(4):27–43, 1989.

Peter M Todd. A connectionist system for exploring melody space. In *Proceedings of the International Computer Music Conference*, pp. 65–65. International Computer Association, 1992.

Stephen Todd and William Latham. *Mutator: a subjective human interface for evolution of computer sculptures*. IBM United Kingdom Scientific Centre, 1991.

Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3 (3):172–186, 2011.

Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1653–1660, 2014.

Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pp. 4790–4798, 2016.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *The Journal of Machine Learning Research*, 9(2579-2605):85, 2008.

Andries Van Der Merwe and Walter Schulze. Music generation with markov models. *IEEE MultiMedia*, 18(3):78–85, 2011.

Dan Ventura. Mere generation: Essential barometer or dated concept. In *Proceedings of the Seventh International Conference on Computational Creativity, ICCC*, 2016.

Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pp. 1096–1103. ACM, 2008.

Pascal Vincent, Hugo Larochelle, Isabelle Lajoie, Yoshua Bengio, and Pierre-Antoine Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

Eric Von Hippel. Lead users: a source of novel product concepts. *Management science*, 32(7):791–805, 1986.

Qing Wang, Pieter Spronck, and Rudolf Tracht. An overview of genetic algorithms applied to control engineering problems. In *Machine Learning and Cybernetics, 2003 International Conference on*, volume 3, pp. 1651–1656. IEEE, 2003.

Renxiao Wang, Ying Gao, and Luhua Lai. Ligbuilder: a multi-purpose program for structure-based drug design. *Molecular modeling annual*, 6(7-8):498–516, 2000.

Zhou Wang, Hamid R Sheikh, and Alan C Bovik. No-reference perceptual quality assessment of jpeg compressed images. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 1, pp. I–I. IEEE, 2002.

Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE transactions on image processing*, 13(4):600–612, 2004.

David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of chemical information and computer sciences*, 28(1):31–36, 1988.

Paul J Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

David White and Richard C Wilson. Generative models for chemical structures. *Journal of chemical information and modeling*, 50(7):1257–1274, 2010.

Thomas Wiegand, Heiko Schwarz, et al. Source coding: Part i of fundamentals of source and video coding. *Foundations and Trends® in Signal Processing*, 4 (1–2):1–222, 2011.

Geraint A Wiggins. A preliminary framework for description, analysis and comparison of creative systems. *Knowledge-Based Systems*, 19(7):449–458, 2006.

Scott A Wildman and Gordon M Crippen. Prediction of physicochemical parameters by atomic contributions. *Journal of chemical information and computer sciences*, 39(5):868–873, 1999.

Ronald J Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural computation*, 1(2):270–280, 1989.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

Stephen Wolfram. *Cellular automata and complexity: collected papers*, volume 1. Addison-Wesley Reading, MA, 1994.

Peter Worth and Susan Stepney. Growing music: musical interpretations of l-systems. In *Workshops on Applications of Evolutionary Computation*, pp. 545–550. Springer, 2005.

Jianxiong Xiao, James Hays, Krista A Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pp. 3485–3492. IEEE, 2010.

Junyuan Xie, Linli Xu, and Enhong Chen. Image denoising and inpainting with deep neural networks. In *Advances in Neural Information Processing Systems*, pp. 341–349, 2012.

Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pp. 2048–2057, 2015.

Raymond Yeh, Chen Chen, Teck Yian Lim, Mark Hasegawa-Johnson, and Minh N Do. Semantic image inpainting with perceptual and contextual losses. *arXiv preprint arXiv:1607.07539*, 2016.

Hiroyuki Yoshikawa. General design theory and a cad system. *Man-Machine communication in CAD/CAM*, 1981.

Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaolei Huang, Xiaogang Wang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. *arXiv preprint arXiv:1612.03242*, 2016.

Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pp. 649–657, 2015.

Yan Zhang, Mete Ozay, Zhun Sun, and Takayuki Okatani. Information potential auto-encoders. *arXiv preprint arXiv:1706.04635*, 2017.

Jun-Yan Zhu, Philipp Krähenbühl, Eli Shechtman, and Alexei A Efros. Generative visual manipulation on the natural image manifold. In *European Conference on Computer Vision*, pp. 597–613. Springer, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint arXiv:1703.10593*, 2017.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Réseaux profonds génératifs pour la génération de nouveauté : fondations, métriques et expériences

Mots clés : Apprentissage automatique, Réseaux profonds, Créativité, Apprentissage non supervisé de représentations, Modèles generatifs, Conception

Résumé : Des avancées significatives sur les réseaux de neurones profonds ont récemment permis le développement de technologies importantes comme les voitures autonomes et les assistants personnels intelligents basés sur la commande vocale. La plupart des succès en apprentissage profond concernent la prédiction, alors que les percées initiales viennent des modèles génératifs. Actuellement, même s'il existe des outils puissants dans la littérature des modèles génératifs basés sur les réseaux profonds, ces techniques sont essentiellement utilisées pour la prédiction ou pour générer des objets connus (i.e., des images de haute qualité qui appartiennent à des classes connues) : un objet généré qui est à priori inconnu est considéré comme une erreur (Salimans et al., 2016) ou comme un objet fallacieux (Bengio et al., 2013b). En d'autres termes, quand la prédiction est considérée comme le seul objectif possible, la nouveauté est vue comme une erreur - que les chercheurs ont essayé d'éliminer au maximum. Cette thèse défends le point de vue que, plutôt que d'éliminer ces nouveautés, on devrait les étudier et étudier le potentiel génératif des réseaux neuronaux pour créer de la nouveauté utile - particulièrement sachant l'importance économique et sociétale de la création d'objets nouveaux dans les sociétés contemporaines. Cette thèse a pour objectif d'étudier la génération de la nouveauté et sa relation avec les modèles de connaissance produits par les réseaux neurones profonds génératifs. Notre première contribution est la démonstration de l'importance des représentations et leur impact sur le type de nouveautés qui peuvent être générées : une conséquence clé est qu'un agent créatif a besoin de re-représenter les objets connus et utiliser cette représentation pour générer des objets nouveaux. Ensuite, on démontre que les fonctions objectives traditionnelles utilisées dans la théorie de l'apprentissage statistique, comme le maximum de vraisemblance, ne sont pas nécessairement les plus adaptées pour étudier la génération de nouveauté. On propose plusieurs alternatives à un niveau conceptuel. Un deuxième résultat clé est la confirmation que les modèles actuels - qui utilisent les fonctions objectives traditionnelles - peuvent en effet générer des objets inconnus. Cela montre que même si les fonctions objectives comme le maximum de vraisemblance s'efforcent à éliminer la nouveauté, les implémentations en pratique échouent à le faire. A travers une série d'expérimentations, on étudie le comportement de ces modèles ainsi que les objets qu'ils génèrent. En particulier, on propose une nouvelle tâche et des métriques pour la sélection de bons modèles génératifs pour la génération de la nouveauté. Finalement, la thèse conclue avec une série d'expérimentations qui clarifie les caractéristiques des modèles qui génèrent de la nouveauté. Les expériences montrent que la sparsité, le niveaux du niveau de corruption et la restriction de la capacité des modèles tuent la nouveauté et que les modèles qui arrivent à reconnaitre des objets nouveaux arrivent généralement aussi à générer de la nouveauté.

Title : Deep generative neural networks for novelty generation: a foundational framework, metrics and experiments

Keywords : Machine learning, Deep learning, Creativity, Unsupervised representation learning, Generative models, Design

Abstract : In recent years, significant advances made in deep neural networks enabled the creation of groundbreaking technologies such as self-driving cars and voice-enabled personal assistants. Almost all successes of deep neural networks are about prediction, whereas the initial breakthroughs came from generative models. Today, although we have very powerful deep generative modeling techniques, these techniques are essentially being used for prediction or for generating known objects (i.e., good quality images of known classes): any generated object that is a priori unknown is considered as a failure mode (Salimans et al., 2016) or as spurious (Bengio et al., 2013b). In other words, when prediction seems to be the only possible objective, novelty is seen as an error that researchers have been trying hard to eliminate. This thesis defends the point of view that, instead of trying to eliminate these novelties, we should study them and the generative potential of deep nets to create useful novelty, especially given the economic and societal importance of creating new objects in contemporary societies. The thesis sets out to study novelty generation in relationship with data-driven knowledge models produced by deep generative neural networks. Our first key contribution is the clarification of the importance of representations and their impact on the kind of novelties that can be generated: a key consequence is that a creative agent might need to rerepresent known objects to access various kinds of novelty. We then demonstrate that traditional objective functions of statistical learning theory, such as maximum likelihood, are not necessarily the best theoretical framework for studying novelty generation. We propose several other alternatives at the conceptual level. A second key result is the confirmation that current models, with traditional objective functions, can indeed generate unknown objects. This also shows that even though objectives like maximum likelihood are designed to eliminate novelty, practical implementations do generate novelty. Through a series of experiments, we study the behavior of these models and the novelty they generate. In particular, we propose a new task setup and metrics for selecting good generative models. Finally, the thesis concludes with a series of experiments clarifying the characteristics of models that can exhibit novelty. Experiments show that sparsity, noise level, and restricting the capacity of the net eliminates novelty and that models that are better at recognizing novelty are also good at generating novelty.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France