An empirical study about the use of Generative Adversarial Networks for text generation

Iván Vallés Pérez

June 13, 2018

Advisors

Dr. D. Anselmo Peñas-Padilla[1] Dr. D. Emilio Soria-Olivas[2]

[1]Escuela Técnica Superior de Ingeniería Informática
Universidad Nacional de Estudios a Distancia

[2]Escola Tècnica Superior d'Enginyeria
Universitat de València

Summary

GAN (*Generative Adversarial Networks*) define a new research line in the generative modelling field. This new paradigm showed impressive results in the computer vision field when they were applied to generate new images from a real data set. Some studies reported results whose generations are clearly indistinguishable from real images to the human eye [1].

Despite that, they have not been broadly applied to generate discrete sequences (e.g. text). One of the most reported issues when generating text using generative adversarial networks is the difficulty that they have of dealing with discrete generations which, indeed, is the nature of text.

The goal of this project is to study how *GANs* can be applied to generate free text and which are the advantages and disadvantages over other common approaches. The best results obtained have been properly reported and quantified.

Contents

Chapter 1

Introduction

1.1 Motivation

The free text generation has had a broad spectrum of applications in the recent years [2, 3, 4, 5, 6]. Rewording assistants, keyword generation and data augmentation are only a few examples.

The use of state of the art techniques such as *Generative Adversarial Networks* for this task can make it possible to push the development barrier.

Some authors, including Ian Goodfellow [1], claimed that this algorithm is not well-suited to perform this task [7]. After a conscientious research, we have found that only a tiny portion of the existing studies report results related to text generation using *Generative Adversarial Networks* and that is why we would like to push the algorithm to its limits in order to study what are the potential lacks and capabilities of it.

1.2 Hypotheses and research questions

This work aims to study the possibility of generating text with the last state of the art *Generative Adversarial Networks* (named below as *GAN*). Complex architectures combining *Recurrent Neural Networks* and *Convolutional Neural Networks* in a *GAN* setting might be powerful enough to generate *pseudo-discrete* generations like text sequences. Below, a set of research questions are enumerated. They are intended to be empirically tested in this work.

- Is there any neural network architecture that performs well for generating text with *GANs*?
- Is it possible to achieve enough level of quality in the generated texts so that the model can be used in a production environment?

[1]Research Scientist currently working in Google Brain and inventor of *Generative Adversarial Networks*

- If the best model achieved is not performing well enough, which are the potential causes?
- Why some architectures performing better than other ones?
- Is the quality of the original dataset affecting on the performance of the algorithm?

1.3 Objectives

The main objective of this work is to exhaust all the possibilities when trying to generate text using *Generative Adversarial Networks*, avoiding using non gradient-based optimization methods typically used in reinforcement learning. The reasons why non gradient-based optimization methods are non-desirable are described below.

- *Generative Adversarial Networks* are a very recent family of algorithms which showed in several studies a high probability of diverging [8, 9].
- *Reinforcement learning training algorithms* are much more complex because they do not have information about the direction of the gradients; i.e. it is more difficult to estimate the changes to be done to the parameters of the network in order to decrease the error.
- The fact of not using a gradient-based approach is a good chance to check how powerful are the neural networks. To succeed, they will have to approximate a sampling over a the probability distribution of the next character of a sequence of text given the previous one.

Below, a set of specific objectives that are intended to be reached during the work are summarized.

- Learning the nuts and bolts of the *Generative Adversarial Networks* algorithms is the most important goal of this work, so that it is possible to be comfortable with these new techniques.
- A fully-differentiable architecture is intended to be designed in order to be able to apply gradient-based optimization methods.
- The generated pieces of text are intended to be generated at character level instead of at word level, because that way there is no possibility of obtaining good results by chance. In addition, a character level approach would lead to a much more powerful model because the cardinality of the set of possible symbols is much smaller. In this case, the network can be able to generate new words (meaning that they have not been shown to the algorithm in the training phase).

- Sentences showing morphologic (correctly generated words), syntactic (correct combinations of words) and semantic (sentences with meaning) sense are intended to be achieved (in increasing order of importance) in order to consider them as successful generations.
- A static (non-diverging) *Generative Adversarial Network* is planned to be implemented. As stated before, these algorithms showed difficulties when dealing with discrete distributions.
- Some general-use deep learning *tricks of the trade* are intended to be discovered in the iterative process of the neural network architecture research.

The general take away of this effort will be an empirical evidence proving the functioning or non-functioning of these algorithms for the described task.

1.4 Structure

This work is structured as follows: the background chapter will give an overview of the deep learning field so that the reader can understand better the current situation. In the methods chapter, fully working modifications of the *GAN* algorithm are explained in detail: *WGAN* and *WGAN-GP*; they will be used for generating text. The proposal chapter explains in detail the nuts and bolts of the algorithms and architectures that have performed better. The experimental setting and results chapter reviews the methodology that has been followed, the metrics that have been used to measure the performance, the generations and the performance level achieved by the proposed algorithms. Finally a conclusions and next steps chapters have been added in order to give a big picture of the whole work, provide answers to the research questions and give the reader an easy way of continuing this research.

Chapter 2

Background

2.1 Neural Networks and artificial intelligence

Ancient references

Human beings, since ancient times, have dreamed about building intelligent machines able to help or imitate us with plenty of different purposes.

The first existing records of our desire to emulate life date from the ancient history, *circa* 384-322 BCE, when the Greek philosopher Aristotle dreamed of automation as a way of making machines work in order to let the humans enjoy leisure [10]. *Circa* 10-70 AD *Hero of Alexandria* wrangled about automatons which would imitate the movements and behaviours of some animals. In the 9-th century, the *Book of Ingenious Devices* was published by *Banu Musa*, which described and presented schemas about different mechanical devices including automatons. Going forward to the Middle Ages, in the 13-th century the scientific, theologist and philosopher *Albertus Magnus* designed several automatons, among which are the *talking head* and an *iron butler* able to do some housework. In the Renaissance, the work of *Leonardo da Vinci* stands out. He designed a couple of automatons, one of them was humanoid [10]. The early-modern and modern history represented the golden ages for the automatons. *Pierre Jaquet-Droz*, in the 18-th century, represented an icon of the history of automation. He built several automatons, *the musician*, *the draughtsman* and *the writer* are the most famous among them. These automatons were built with up to six thousand pieces and they are incredibly complex. They are currently considered the predecessors of the modern computing. Despite the complexity of all the listed machinery, there was no intelligence involved in them as we understand it right now; they were very complex mechanical machines designed to surprise the audience in the shows.

There are also a high number of fiction stories that can be found in our historical bibliography. For instance, one of the first fictions references something that resembles to artificial intelligence appears in *The City of Brass*, a tale of *The One Thousand and One Nights* book of 1706 (in its English version). Some beings described like humanoid robots appear in the middle of the story. In the 19-th century, *E. T. A Hoffman* published a book of short stories called *The Night Pieces*, which contained a story called *The Sandman*. It deals about a student with a complicated childhood who falls in love with an automaton being convinced that it was real. When he realizes it was a machine he runs mad. The 20-th century represented the most prolific era when it comes to science fiction. *Karel Capek* and *Isaac Asimov*, icons of the fictional literature and almost known by everybody, are nowadays considered the parents of the robotics. *Karel Capek* was the first person introducing the word *robot* (which seems to come from some Czech word meaning *work*) in his book *Rossum's Universal Robots* in 1920. He played with the ideas of artificial intelligent beings able to think by themselves. They were created to replace humans in the hard work. Finally, the story concludes with these creatures fighting against the humanity. *Isaac Asimov*, on the other hand, wrote the *Foundation Series*, which represented a keystone on the philosophy about robotics. He defined the *Three Laws of Robotics* (depicted below), which were a set of principles intended as a fundamental framework to support the robots behavior; they formed the basis of the actual robot design thinking.

1. *A robot may not injure a human being or, through inaction, allow a human being to come to harm.*

2. *A robot must obey the orders given it by human beings except where such orders would conflict with the First Law.*

3. *A robot must protect its own existence as long as such protection does not conflict with the First or Second Laws.*

Modern artificial intelligence

The *McCullouch-Pitts neuron* [11] (see figure 2.1) is considered one of the pioneer models in the *AI* field. It consisted of a simple mathematical equation that tried to mimic the way biological neurons work. This model consisted of a set of parameters that weighted the input values and applied a threshold in order to output a binary value as shown in the equation 2.1. The weights of the neuron w_j were adjusted manually by an operator in order to produce the predicted output $\hat{y}$ given the inputs x_j. This kind of model allowed to compute logical functions and founded the basis of artificial neural networks.

$$\hat{y} = sign\left(\sum_{j=0}^{m} w_j x_j\right) \tag{2.1}$$

Alan Turing —the father of modern computer science— through his deep study in mathematics and formal logic, suggested that a machine could be programmed to do complex mathematical deductions through binary encoding [12]. This was the begining of the era in which the artificial intelligence would

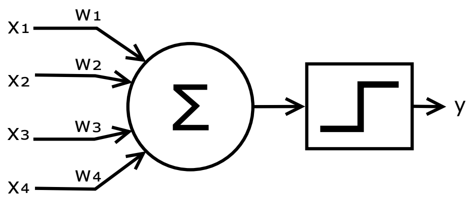


Figure 2.1: McCullouch-Pitts neuron ($m = 4$ inputs)

be materialized. After the *Turing's* success defeating the German *Enigma* cryptographic system [13], in 1950, he wrote a seminal paper in which he discussed a philosophical question: *"Can machines think?"* [14]. He introduced a test to identify if a machine is intelligent enough to generate human-like responses in text format. It is currently known as *the Turing test* and consists of set of a human-computer conversations in which the human has to determine if the answers he is getting from the computer have been generated by another human or by the machine. Turing suggested that if the human is beaten the 70% of the times after 5 minutes conversations, the machine would be considered intelligent. This study has been strongly considered and served as inspiration for most of the current scientific community around artificial intelligence.

In 1956 the artificial intelligence term was coined by *John McCarthy*, *Nathaniel Rochester* and *Claude Shannon* in the *Dartmouth Summer Research Project on Artificial Intelligence*: a 2 month workshop that was organized by the mentioned authors in order to found the artificial intelligence as an academic discipline. There were 11 invited scientists among which stand out: *Marvin Minsky*, *John Nash* and *Ray Solomonoff.*

Figure 2.2: *Marvin Minsky*, *John McCarthy*, *Claude Shannon*, *Ray Solomonoff* and other scientists attending the *Dartmouth Summer Research Project on Artificial Intelligence*

The years after the Dartmouth workshop there was a general optimism feeling surrounding the artificial intelligence field which was accompanied with a general feeling of hype. In 1974, *Sir James Lighthill* published a paper currently known as *the Lighthill report* [15] in which he criticized in a very pessimistic way the ideas about the research in some of the functional areas in artificial intelligence. This publication, along with the general oversized expectation of the scientific community regarding the future of the field formed the basis for the British and the US government to stop dedicating funds to the artificial intelligence exploratory research [16], leading to what it is currently known as **the first AI winter**.

After several years, in the early 1980s, some industries started using artificial intelligence systems again, but they changed its name to *expert systems*. The AI market started growing newly and it motivated the research community to continue exploring and designing new algorithms. The most remarkable invention was the successfully application of the *backpropagation* algorithm to the neural networks by *David Rumelhart*, *Geoffrey Hinton*, *et al.* [17]. It was an algorithm that consisted of back-propagating the derivatives of a given error function in order to adjust the synaptic weights accordingly, allowing training deeper neural networks in an easier way.

All these consecutive successes propitiated the hype to grow again and in 1985 several specialized companies surrounding the artificial intelligence field were founded. Some of them, specially *Lisp Machines Inc.*, started building hardware specifically designed for the research needs. All this generated an economic bubble which became in a market collapse leading to **the second AI Winter**.

1990s started with a renewed mind in the artificial intelligence field by being more focused on applications. Machine learning proved to be effective for many problems [18]. It was motivated by the increasingly computation power (as predicted by *Gordon Moore* in 1965 [19]). There appeared a high number of interesting applications in several industries like finance and logistics [20, 21, 22, 23], psychology [24, 25, 26, 27] and medical diagnosis [28, 29, 30, 31], and others [32, 33, 34, 35]. Maybe the most remarkable success in the field was achieved in 1997, when the *IBM's Deep Blue* supercomputer defeated the world chess champion *Garry Kasparov* [36].

The 21-th century has represented an explosion of the field. Expert systems have suffered another rebranding and now they are known as **deep learning** (it will be described in the next section). Reinforcement Learning has evolved until astonishing limits; the success in *the game of Go* starring *AlphaGo Zero* [37] has represented an incredible progress for the surrounding scientific community. *AlphaGo Zero*, designed by *David Silver* and his team for *Google Deepmind*, achieved superhuman levels in *the game of Go* and it was the first algorithm able to learn from scratch without being provided with data; instead, it played against itself. The same algorithm has also recently proved to be scalable to other board games like *Chess* or *Shogi* [38].

Artificial intelligence at present

The current artificial intelligence research community is enjoying of one of the most prolific eras. Deep learning is possibly one of the most fruitful sub-fields of artificial intelligence in terms of development and application. The industry is seriously adopting these techniques in their daily developments and each and every imaginable industry has its own niche for artificial intelligence enhancements. The applications are growing at an incredible pace, possibly motivated by the ease of communication provided by the Internet and the new technologies. The community development also takes an important role in this success since the specific average conferences attendance is growing exponentially (e.g. *NIPS* [1]).

However, there is still a part of the community which are resilient to the new wave of algorithms. They think that the artificial intelligence is still on very early stages of development and very far from the application phase. The current state-of-the-art has recently been compared with "alchemy" by people following this current of thought [39], due to a misconceived lack of mathematical support. This topic has been hardly criticized by eminences in the field like *Yann Lecun* and *Yoshua Bengio* [2]. These discussions must be delicately treated so that they will not cause a confusion that lead the community to the third AI winter.

2.2 Deep learning

Deep learning is considered by most of the authors [40] as a subfield of artificial intelligence, as it is shown in the Venn diagram of the figure 2.3. Figure 2.4 shows a summary of the most important achievements in the deep learning community.

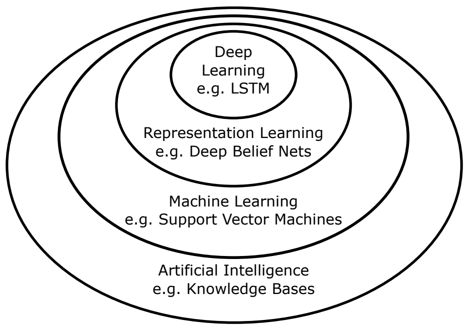


Figure 2.3: Venn diagram locating deep learning as a subfield of other disciplines, being artificial intelligence the most general one.

[1]Neural Information Processing Systems
[2]`https://www.facebook.com/yann.lecun/posts/10154938130592143`

History of deep learning

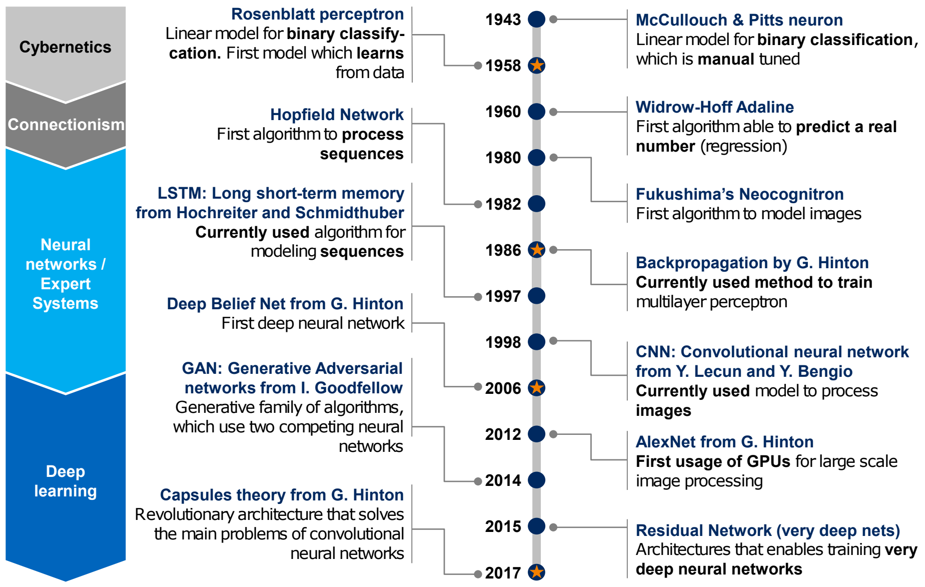

Cybernetics
Connectionism
Neural networks / Expert Systems
Deep learning
1943
1958
1960
1980
1982
1986
1997
1998
2006
2012
2014
2015
2017
Rosenblatt perceptron
Linear model for binary classification. First model which learns from data
Hopfield Network
First algorithm to process sequences
LSTM: Long short-term memory from Hochreiter and Schmidthuber
Currently used algorithm for modeling sequences
Deep Belief Net from G. Hinton
First deep neural network
GAN: Generative Adversarial networks from I. Goodfellow
Generative family of algorithms, which use two competing neural networks
Capsules theory from G. Hinton
Revolutionary architecture that solves the main problems of convolutional neural networks
McCullouch & Pitts neuron
Linear model for binary classification, which is manual tuned
Widrow-Hoff Adaline
First algorithm able to predict a real number (regression)
Fukushima's Neocognitron
First algorithm to model images
Backpropagation by G. Hinton
Currently used method to train multilayer perceptron
CNN: Convolutional neural network from Y. Lecun and Y. Bengio
Currently used model to process images
AlexNet from G. Hinton
First usage of GPUs for large scale image processing
Residual Network (very deep nets)
Architectures that enables training very deep neural networks

Figure 2.4: Timeline showing the different breakthroughs and milestones in the history of deep learning. The field has been re-branded several times, as it is shown in the left diagram.

As discussed in section 2.1, *McCullouch-Pitts* [11] designed the first neural-based system in 1943 but it lacked of training algorithm. Instead, it was tuned manually by an operator. Several years later, in 1958 *Frank Rosenblatt* proposed *the Perceptron*, which share the same idea that *McCulloch-Pitts* proposed (see figure 2.1) but accompanied with a learning algorithm that allowed the system to learn automatically (i.e. adjust its parameters) from data [41]. It consisted of a simple and effective formula that is shown in the equation 2.2, for all the variables $0 \leq j \leq M$ and all the training instances $0 \leq i \leq N$, where $x^{(i,j)}$ represents the input value of the variable j of the instance i, $\hat{y}$ is the actual prediction, y is the desired prediction and t represents the current iterative state. The *Rosenblatt's Perceptron* is considered the first neural network and was intended to solve binary classification algorithms (by applying a sign function on the output of the system).

$$w^{(j)}(t+1) = w^{(j)}(t) + (y^{(j)} - \hat{y}^{(j)}(t))x^{(j)} \tag{2.2}$$

Two years afterwards, in 1960, *Bernard Widrow* and his doctoral student *Ted Hoff* presented the *ADALINE* algorithm at *Stanford University* [42], which stands for *Adaptive Linear Neuron*. Its structure is quite similar to the *Rosenblatt's Perceptron* but the learning algorithm is different. The only differences in the inference phase of the algorithm are that it includes a bias θ and that the sign function is no longer applied (see equation 2.3). The learning phase is done by gradient descent: considering the Mean Squared Error measured in the output of the algorithm ($E = \frac{1}{2}\sum_{i=1}^{N}(y^{(i)} - \hat{y}^{(i)})^2$ where the 1/2 term has been included there to simplify the subsequent derivative. The derivative of the error with respect to the weights is calculated as it is shown in equation 2.4. Then, the weights can be updated according to the equation 2.5, where λ represents the learning rate, a hyperparameter that controls how fast the updates of the parameters are made. For the sake of formulation simplicity, the bias (θ) can be considered as another weight simply by renaming it as $w^{(j=0)}$ with $x^{(j=0)} = 1$.

$$\hat{y}^{(i)} = \sum_{j=0}^{M} w^{(j)} x^{(i,j)} + \theta \tag{2.3}$$

$$\frac{dE}{dw^{(j)}} = \frac{1}{N}\sum_{i=1}^{N}\left(x^{(i,j)} \cdot (\hat{y}^{(i)} - y^{(i)})\right) \tag{2.4}$$

$$w^{(j)}(t+1) = w^{(j)}(t) - \lambda \cdot \frac{dE}{dw^{(j)}} \tag{2.5}$$

That was the first time that gradient descent was used in this context, as it will be explained below, it formed the base for more complex algorithms.

Some years later, the *first AI winter* moment arrived. The neural networks field suffered because of the artificial intelligence general dampening and because some important limitations of the current algorithms were published by *Marvin Minsky* [43] (the existing algorithms were unable to solve non linearly separable problems like the *XOR* function).

In 1980, once the confidence in the field started being restored, *Kunihiko Fukushima* published his work on a new algorithm able to model images: *the Neocognitron* [44]. It was based on complex operations that occur in the primary cortex of the human brain. *Fukushima* proved that the algorithm was successful at recognizing handwriten digits. Its operation is similar to a regular *convolutional neural network* and it implements similar properties: weights sharing and translation equivariance. This algorithm served as inspiration of modern *convolutional neural networks*.

John Hopfield, in 1982 published a new algorithm currently known as *Hopfield Network* that was the first model which incorporated a memory cell that allowed to model time-sequences [45]. It was composed of binary units in which the values were determined by the application of a threshold. The *Hopfield network* added connections between neurons in the same layer but with some constraints: (i) a unit cannot be connected to itself, (ii) the connections in a given layer must be symmetric to assure convergence.

In 1986, *David Rumelhart* and *Geoffrey Hinton* published maybe the most important paper in the field [17]. In it, they explained the most currently used method for training all kinds of multilayer neural networks: *the Backpropagation Algorithm.* It consists of two phases.

- Forward pass: this is the inference phase in which the input values flow through the network, from the very first layer (input) to the last layer (output), in order to calculate the output value(s).

- Backward pass: in this phase, the value(s) computed in the forward pass are compared with the desired value(s) (i.e. the correct value that should have been outputted by the network) through an error metric. Then the error signal (i.e. the gradient of the error) is back-propagated using the chain-rule, from the last layer to the first layer in order to adjust the weights of the network, typically through gradient descent.

This study represented a truly breakthrough because until that moment the only possible way of training *multilayer perceptrons* was by manually fixing all but one hidden layer input weights (that specific kind of networks were called "feature analyzers" [17]). *Hinton*, *et al.* explained that since that moment, the *perceptron-based* algorithms were not *learning representations* [3]. The authors claimed that the algorithms trained with *backpropagation* would "construct appropiate internal representations" to decrease the error when predicting the desired output.

The authors proposed using the *sigmoid function* (defined in equation 2.6) as activation function to the network instead of the sign function (which wouldn't be derivable). Finally, 2 methods for training the perceptrons were proposed: the *online* method, in which the weights of the networks are iteratively updated after each input-output case, and the *batch* method, in which the gradients of the error with respect to the weights of the network are accumulated and the update is performed at the end of the epoch (i.e. when all the input-output cases have been evaluated).

$$\sigma(x^{(i)}) = \frac{1}{1 + e^{x^{(i)}}} \tag{2.6}$$

The algorithm was presented with some constraints: (i) connections from higher level layers to lower level ones are forbidden while connections which skip layers are totally permitted, (ii) the whole function must have bounded derivatives in order to be able to back-propagate the error, (iii) the layers must be randomly initialized to different weights in order to break symmetrical weights between layers (i.e. the weights being updated to the same values in different layers in each backward pass). The authors warn that the algorithm does not theoretically guarantee to raise the global optimum, though in the practice, it was only observed when the network was configured with just the number of necessary weights to solve a given problem and it was easily solved by adding more units to the network.

The same year, *Paul Smolensky* [46] invented the *Harmoniunm* (later called *Restricted Boltzmann Machine* by *Geoffrey Hinton* and collaborators). It consists of a powerful neural network that given a set of input variables, learns

[3]that was the begining of the representation learning subfield

unsupervised representations which can be useful for disentangling abstract information. The architecture consists of a symmetrical bipartite bidirectional graph with shared weights, as shown in the figure 2.5. The network is trained to learn a representation from which it is possible to recover the input probability distribution $P(x|a) \approx P(x)$ by minimizing the *Kullback-Leibler* divergence [47] between the input x and its reconstruction $\hat{x}$: $D_{\mathrm{KL}}(x\|\hat{x}) = -\sum_j x^{(j)} \log \frac{\hat{x}^{(j)}}{x^{(j)}}$,

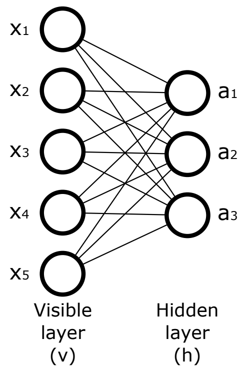


Figure 2.5: *Restricted Boltzmann Machine* with 5 inputs (x) and 3 outputs (a)

In 1997, *Sepp Hochreiter* and *Jürgen Schmidhuber* proposed an evolution of the *Hopfield network* which claimed to handle the long term dependencies much better than previous algorithms and to be much more stable than the existing recurrent neural networks: the *Long-Short Term Memory* (*LSTM* in advance) [48]. It belongs to the family of the so-called recurrent neural networks (*RNNs*). Instead of adding a recurrent connection to each neuron (connection from the output of the neuron pointing to its input), its entrails are much more complicated. Further details on the *LSTM* cell are provided in the figure 2.6. The *LSTM* is the most broadly used recurrent cell nowadays [49, 50].

The next year, in 1998, *Yann Lecun*, *Yoshua Bengio* and collaborators published the *LeNet-5* architecture: a 7-layer *convolutional neural network* (apart from the input) which proved to be successful recognizing handwritten digits [51]. This network was applied in some industries in order to help automating specific tasks by recognizing handwritten digits. The architecture of the network is summarized in the figure 2.7.

In 2006, *Geoffrey Hinton* starred in a new breakthrough which would change the name of the field to **Deep Learning**. He stacked several *restricted Boltzmann machines* allowing training deeper representations with dimensionality reduction purposes; they called this approach *Deep Belief Networks* (*DBNs*). They empirically proved that given a data set, it was much easier to train a *deep autoencoder* (a network with a *bottleneck* that tries to reproduce its input in the output producing a compressed representation) using the greedy layer-wise pretraining through a *DBN*. The approach consisted of learning feature representations from the input using the *RBM* approach, and then stacking another *RBM* on the output of the previous one (already trained) in order to use its

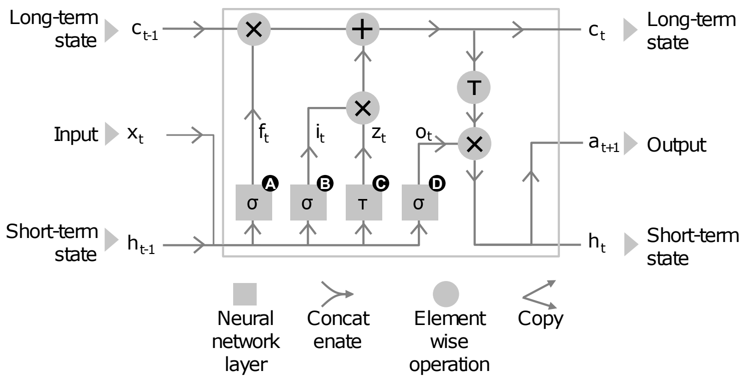


Figure 2.6: *LSTM* cell structure. c_t and h_t represent the long-term and short-term states that the network uses as memory, respectively. Its operation is based on 3 gates and an output unit. A is a layer that acts as forget gate, which is responsible for erasing memory which will no longer be used. The layer B is the input gate and controls how much input goes through the long-term line. The layer C controls the new contribution to the cell state that, in conjunction with the layer B form the memory addition system. The layer D is the output gate, which controls which values are outputted. In the diagram, σ stands for the logistic function and τ for the hyperbolic tangent.

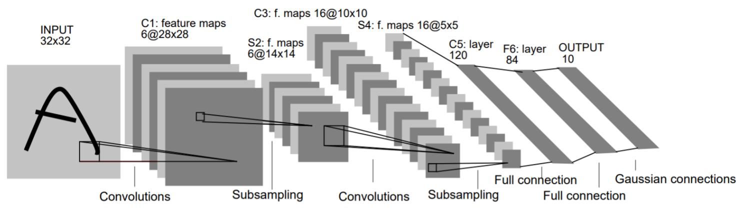


Figure 2.7: *Lenet-5* architecture, as appeared in the original publication. They used two blocks containing a 5x5 convolutions layer (in *same mode*) followed by a pooling layer and 3 fully connected layers at the end.

output as the input of the current one (see figure 2.8). As a very last step, the authors recommend to stack all the pretrained layers unrolling them in an encoder-decoder setting and perform a fine-tuning using the *backpropagation* algorithm.

The next year *Yoshua Bengio* and collaborators showed in the *NIPS conference* that the *Deep Belief Network* could be used for *pretraining* deep neural networks in order to use the final weights as an initialization for a determined supervised task [52]. They empirically proved in the article that applying the *greedy layer-wise unsupervised pretraining* using *DBNs* alleviated the complex optimization problem of deep networks: the difficulty of convergence to a good local optimum [53]. This success enabled the researchers to train deeper neural networks and the field started being known as deep learning.

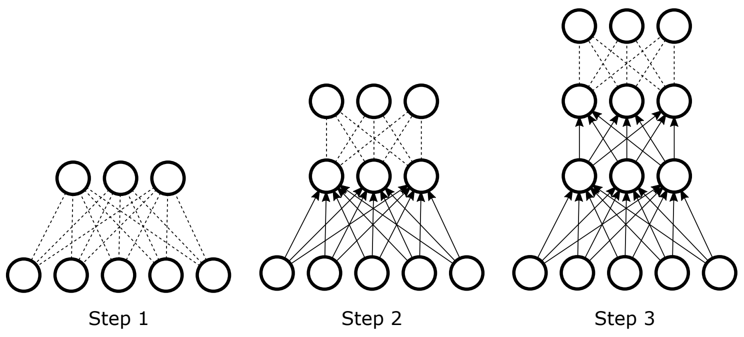


Figure 2.8: *Deep Belief Network* example of architecture in which three layers of feature detectors have been pretrained by stacking *RBMs.* The dashed connections represent the undirected connections between the neurons in the *RBM* which is being trained in every step, while the solid arrows represent the already trained layers that are used to calculate the input of the next *RBM* to be trained.

Modern deep learning

In the 21-th century the deep learning has become a hot topic for all the scientific community. There are a lot of research branches studying the field and inventing new algorithms everyday all around the world.

In 2012, *Alex Krizhevsky*, student of *Geofrey Hinton*, and his team presented *AlexNet* [54] a very exhaustive *convolutional neural network* that won the *ImageNet* competition (consisting on classifying over 15 milion of labeled images belonging to nearly 22000 categories). They employed *Rectifier Linear Units* (*ReLU*) as activation functions ($y = max(0, 1)$) in the network architecture, which showed much faster convergence than the sigmoid and hyperbolic tangent. This new activation function also showed to be effective favoring sparse connectivities. Since this moment, *ReLU* is considered the *de facto* activation. They also specify a bunch of tricks that were used in the network design and the training methodology that improved the overall model. The most remarkable one is the use of *Dropout*, a regularization technique consisting of adding a probability to each neuron to be deactivated in each iteration. This method was recently published also by *Geoffrey Hinton's* team [55, 56].

Another remarkable success was achieved by the *Google* researchers Tomas Mikolov et al. in 2013. They invented *Word2Vec*: a new method that used embedding matrices to allow vectorizing the words of a given language so they learn part of the semantics in an unsupervised manner [57]. It consisted of building a model that was able to determine if a word belonged to a context or not using a latent vector associated with each of the words. Once the model converges, the vectors associated with each word encode very rich information about the meaning of them. As a curiosity, it is possible to make simple algebraic operations with the learned representations (e.g. $\vec{w}_{king} - \vec{w}_{man} + \vec{w}_{woman}$ would produce a vector which is very similar to $\vec{w}_{queen}$).

Ian GoodFellow presented in 2014 the *Generative Adversarial Networks* [8], a new kind of algorithm for approaching generative modeling. It was based on two competing neural networks which played in a 2 role game: the counterfeiter (so-called Generator) trying to make fake money and the police (so-called Discriminator) trying to discriminate between fake and real money. These algorithms proved to be very effective generating images but they were extremely difficult to train, specially because in most of the cases one of the two models become too strong compared with the other and the system diverges [58].

The list of recent remarkable achievements is very extensive so only the ones which are most related with this work have been mentioned.

2.3 Generative modeling

One of the most trending topics in the current state of the art of deep learning is generative modeling. These techniques belong to the unsupervised learning branch since they don't need labeled data to be executed. The general approach consists of collecting a big amount of data in some domain (they can be of any type: images, text, audio, video, structured data, graphs, etc.) and train a model which will generate data like them.

There are different types of generative models. They share some properties and differ in other ones but there is not a clear way of deciding which one is better than the other; each of them has its own advantages and disadvantages. In this section the three most popular approaches will be covered: *autoregressive models (Fully Visible Belief Networks, FVBN)*, *Variational Autoencoders* and *Generative Adversarial Networks* (see figure 2.9). Different generative modeling algorithms will be discussed from the deep learning perspective.

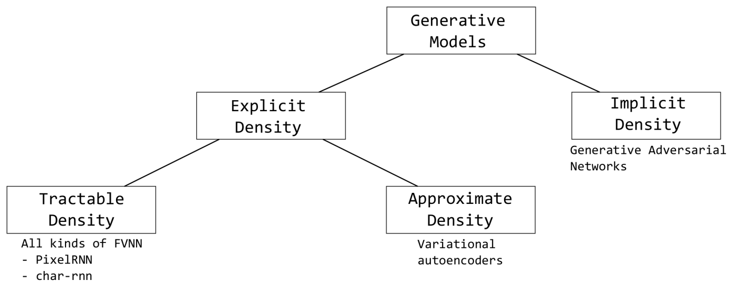


Figure 2.9: Generative Modeling Taxonomy, borrowed and simplified from [59].

The generative models are intended to "understand" the data that they are trained on, rather than memorizing them. One of the reasons why this assumption can be taken is that the models have much less parameters than the size of the data, so no data memorization can be considered if the model successfully predicts good quality samples

Generally speaking, generative models start learning from scratch. They have to learn everything related to the structure and variability of the data they are designed to generate. In the case of the images, for instance, they have to learn different textures, different backgrounds, the fact that nearby pixels have a high likelihood of having similar colors, etc. In the case of character-level text generation, they will learn from the most basic concepts like starting the sentences with a capital letter, inserting a space after a comma or separate the concepts in words, which will be separated with a space, the correctly use of tenses, the use of singular and plural forms, or even the use of the passive voice.

Mathematically, considering a set of examples $x^{(1)}, x^{(2)}, ...x^{(N)}$ sampled from a real distribution $p_{data}(x)$, the goal of the generator is to find a set of parameters θ such that $p^{\theta}_{model}(x) \approx p_{data}(x)$ given a divergence metric (for example the KL Divergence [47] or the JS Divergence [60]). The figure 2.10 shows a conceptual example of this process.

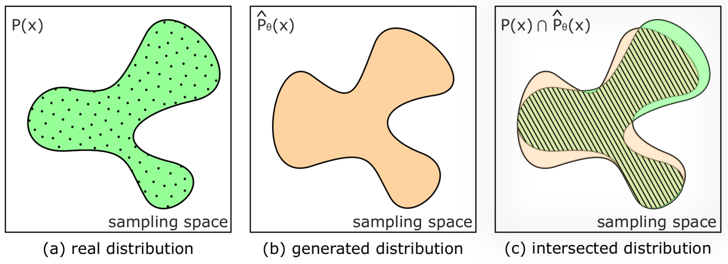


Figure 2.10: Given a real distribution (a), in which the dots represent the empirical distribution that the model is considering for infering the modeled distribution, the set of parameters θ of a model are adjusted so that $p_\theta(x) \approx p(x)$, producing (b). As it can be seen in (c), the match is not exact. There is a part of the space which matches (stripped pattern), another part in which there is real distribution and the model does not cover (green), and another part of the space the model samples from but, that would not be considered as real samples (red).

The generative modelling is a very complex task that requires of very advanced techniques. That is maybe the main reason why deep learning takes advantage of the situation. The main challenge that appears in the generative models field is the difficulty of correctly measure the success of the models. How is it possible to mathematically determine if a image generator is generating images which look great to a human eye? or how is it possible to measure how natural it feels a generated sentence? Some algorithms (autoregressive models, for instance) directly maximize the likelihood of the data, so that the quality of the generations is directly correlated with the estimated probability. Others instead, do not provide an explicit density function and the quality of the results is much more difficult to assess (this is the case of *GANs*).

2.3.1 Autoregressive models (Fully Visible Belief Networks)

This family of algorithms represent the most simple approach in the generative modeling field. They are based on the idea of generating data given previous samples, i.e. they implement a “next-step prediction” model: $p_{model}(x) = \prod_{i=1}^{t} p_{model}(x^{(i)}|x^{(1)}, ..., x^{(i-1)})$. These models are also known as *Fully-visible belief networks* (*FVBN*) and are typically trained by applying Maximum Likelihood [61, 59]. The density distribution obtained through the application of this algorithm is fully tractable, i.e. the p_{model} density function is known.

These models are very well-suited for discrete data distributions generation since the inference process is simply based on sampling from a modeled probability distribution. They can generate one step at a time: in the case of the images, they can grow the image a bunch of rows and columns in each step; in the case of character-level language generation, they generate a character at a time. That is the reason why these models are very inefficient in time complexity (O(n)). Furthermore, these models are not parallelizable, due to their sequential nature.

An example of autoregressive model for natural language generation at character level is the *char-rnn* model [62, 63]. This model uses a recurrent neural network trained to generate the next character given a sequence. Once it is trained and converges, the inference phase consists of choosing a start character and making chained predictions: for each prediction, the network will output a likelihood for each character, then a sampling given the output probabilities should be made in order to forward the chosen character to the input of the next state. The process is graphically described in figure 2.11

With the aim of providing intuition about the modeling power of this approach, we have trained a shallow *char-rnn* (i.e. one hidden layer) with a window size of 100 characters[4] using 800 randomly chosen books from the Gutenberg[5] project. After achieving convergence, the model has been provided with the first 100 characters of the War of the Worlds book as initialization. The results are shown below.

> ***“No one would have believed in the last years of the nineteenth century that this world was being wat****ched together far out to the thought, ’that will cled it announced for them bloody I have last instant all the strayes as thomass us? This is, and the single War camp, until we proguted it toward the mouncin to lint of the enemy respectful, then the ribonament had been their courts and papers she been ended bent tense freely after good to the eyes to avole. The gather flooded by Wayer a great time home engaged in the*

[4]meaning $P_\theta(x^{(t)}) = P\theta(x^{(t-100)}) \cdot P\theta(x^{(t-99)}) ... P\theta(x^{(t-1)})$, assuming it is a *Markov* process

[5]`https://www.gutenberg.org`

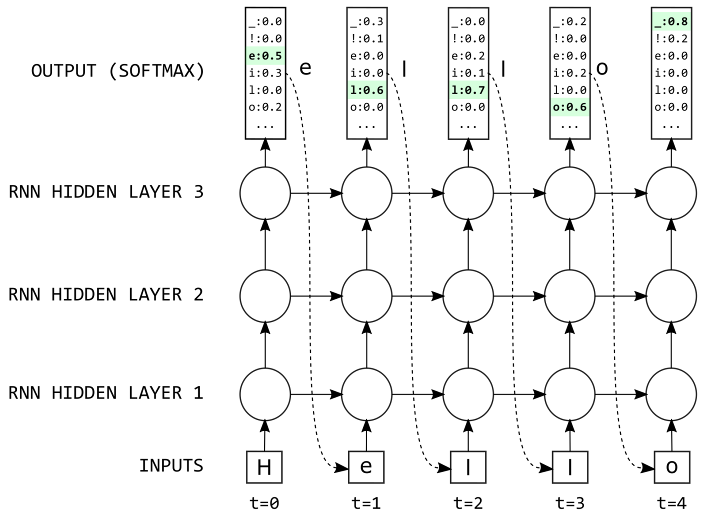


Figure 2.11: *char-rnn* autoregressive architecture for natural language generation in its inference phase using a *multi-rnn* setting. The bubbles represent hidden recurrent cells (e.g. LSTM), the squares are the inputs and outputs and the dashed arrows show the sampling process. For the sake of clarity, the maximum probability has been taken in the output sampling process.

> *exhausting day of the dutie of summers and jangers untourant altogether of mountains. But was the mystery arising half die for some regarding the raider. This top was a Catelumberhand life by the river were business, and other ebbsake and septimum at the campaich, wa, and he showed my brethren."*
>
> \- *char-rnn, 2017*

In the case of the images, PixelRNN is a great example of *FVBN* autoregressive model which was published in 2016. It can be seen an extension of the *char-rnn* to 2 dimensions [64] that proved to achieve successful results in images.

2.3.2 Variational autoencoders

Variational Autoencoders (VAE) are a probabilistic version of traditional autoencoders that allow generating data from them by sampling from a random distribution of latent variables z [65, 66]. These algorithms define an explicit but intractable density function which cannot directly be optimized. Instead, variational inference methods are used as an aproximation to maximize the likelihood of $p(x)$.

Given the probabilistic model structure of figure 2.12, the posterior probability $p(z|x)$ is intended to be computed. Applying Bayes theorem: $p_\theta(z|x^{(i)}) =$

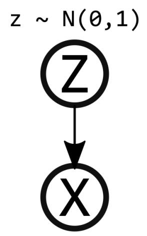


Figure 2.12: VAE as a structured probabilistic model.

$\frac{p_\theta(x^{(i)}|z)\cdot p_\theta(z)}{p_\theta(x^{(i)})} = \frac{p_\theta(x^{(i)},z)}{p_\theta(x^{(i)})}$ where $p_\theta(x^{(i)}) = \int p_\theta(z) \cdot p_\theta(x^{(i)}|z)\delta z$ is intractable, i.e. it is not possible to optimize it directly.

Here is where variational inference takes place. In order to arrive to a solution, $q_\phi(z|x^{(i)}) \approx p_\theta(z|x^{(i)})$ will be found. To achieve it, Kullback-Leibler (D_{KL}) divergence [47] between p and q will be used. The objective of the problem is reduced then to the expressions shown in the equation 2.7

$$\min_{\theta,\phi} D_{KL}\left(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)})\right) \tag{2.7}$$

$$\begin{aligned}
D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)})) &= -\sum_z \left(q_\phi(z|x^{(i)}) \cdot \log\frac{p_\theta(x^{(i)}|z)}{q_\phi(z|x^{(i)})}\right) \\
&= -\sum_z \left(q_\phi(z|x^{(i)}) \cdot \left(\log\frac{p_\theta(x^{(i)},z)}{q_\phi(z|x^{(i)})} - \underbrace{\log(p_\theta(x^{(i)}))}_{\substack{\text{does not}\\ \text{depend on z}}}\right)\right) \\
&= -\sum_z \left(q_\phi(z|x^{(i)}) \cdot \log\left(\frac{p_\theta(x^{(i)},z)}{q_\phi(z|x^{(i)})}\right)\right) ... \\
&\quad + \log(p_\theta(x^{(i)})) \cdot \cancelto{1}{\sum_z \left(q_\phi(z|x^{(i)})\right)}
\end{aligned}$$

$$
\begin{aligned}
log(p_\theta(x^{(i)})) &= D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)})) + \sum_z \left(q_\phi(z|x^{(i)}) \cdot \log \left(\frac{p_\theta(x^{(i)}, z)}{q_\phi(z|x^{(i)})} \right) \right) \\
&= D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)}))... \\
&\quad + \sum_z \left(q_\phi(z|x^{(i)}) \cdot \log \left(\frac{p_\theta(x^{(i)}|z) \cdot p_\theta(z)}{q_\phi(z|x^{(i)})} \right) \right) \\
&= D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)})) + \sum_z \left(q_\phi(z|x^{(i)}) \cdot \log \left(p_\theta(x^{(i)}|z) \right) \right) ... \\
&\quad + \sum_z \left(q_\phi(z|x^{(i)}) \cdot \log \left(\frac{p_\theta(z)}{q_\phi(z|x^{(i)})} \right) \right) \\
&= \underbrace{D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z|x^{(i)}))}_{>0} ... \\
&\quad + \underbrace{\mathbb{E}_z \log p_\theta(x^{(i)}|z) - D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}
\end{aligned}
$$

Where the $\mathcal{L}(x^{(i)}, \theta, \phi)$ represents a lower bound in the optimization function (see equation 2.8) because the KL divergence of equation 2.7 is always greater or equal to zero. As the lower bound expression is tractable and maximizing it would, indeed, maximize $p_0(x^{(i)})$, the optimization problem can be approximated as shown in equation 2.9. Maximizing the lower bound would assure that the log-likelihood of the data is at least as big as its lower bound [66].

The lower bound is basically formed by 2 components: the generation loss and the latent loss. The first one is $\mathbb{E}_z \log p_\theta(x^{(i)}|z)$ and quantifies the quality of the samples at the output of the decoder (the reconstruction error). The second one is $D_{KL}(q_\phi(z|x^{(i)}) \parallel p_\theta(z))$ and keeps the distribution of z close to $p_\theta(z)$.

$$\log(p_0(x^{(i)})) \geq \mathcal{L}(x^{(i,\theta,\phi)}) \tag{2.8}$$

$$\theta, \phi \leftarrow \arg\max_{\theta,\phi} \sum_{i=1}^{N} \left(\mathcal{L}(x^{(i,\theta,\phi)}) \right) \tag{2.9}$$

In terms of neural networks architectures, $q_\phi(z|x^{(i)})$ and $p_\theta(x^{(i)}|z)$ are the encoder and the decoder of the variational autoencoder architecture, respectively, where ϕ and θ are the respective parameters of the networks, x is the input data sample and z is the set of latent variables.

The distribution of the latent variable $P(z)$ is usually chosen as a $\mathcal{N}(0, 1)$. In order to be able to force this constraint while assuring global derivability in all the parts of the architecture, a *reparametrization trick* is used. It consists of predicting the mean and the covariance of the latent vector variables separately in the output of the encoder and then sampling from the distribution defined by those parameters (see figure 2.13). This is different than the classical autoencoder setting in which the encoder directly generates z.

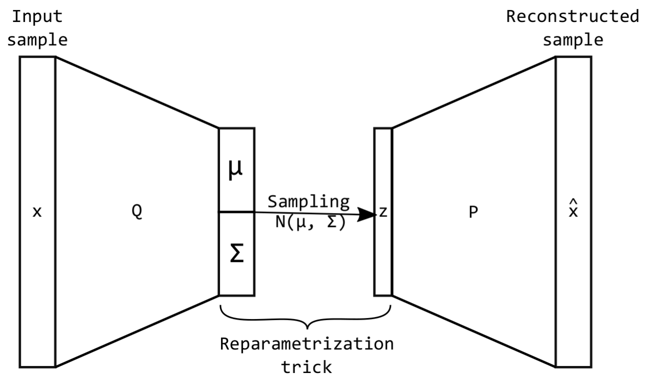


Figure 2.13: VAE neural network architecture and *reparametrization trick*.

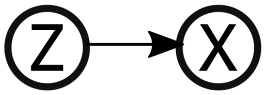


Figure 2.14: GAN as a structured probabilistic model.

2.3.3 Generative adversarial networks

The *Generative Adversarial Network* (*GAN*) are the most recent family of algorithms presented in this section. It has had an enormous support in the research community since it was presented in 2014 [8]: in 4 years, more than 250 versions and enhancements have been published [6].

The objective of the algorithm is to learn to represent an estimate (called p_{model} subsequently) of a given distribution (called p_{data} from now on). This family of algorithms are able to learn the p_{model} distribution implicitly, allowing sampling from it but not providing access to the distribution itself. It has been proven [59] that if infinite amounts of data is provided to a large enough model, the convergence of the algorithm corresponds to recovering p_{data} exactly.

The original algorithm consists of a neural network system built with 2 players: a generator and a discriminator. The whole system is optimized (sometimes in a zero-sum game) until the *Nash equilibrium*[7] is reached (ideally). If any of the networks becomes unexpectedly strong, the system usually diverges. However there are mechanisms to avoid it [59]. There has to be a strength compromise between both networks to assure stability and there are different approaches to control these situations. The whole system is a structured probabilistic model [40] containing latent variables z and observed variables x (see figure 2.14).

The generator takes z as input data and $\theta^{(G)}$ as parameters. This network, being provided with a vector z of random variables (typically 100), is in charge

[6]The GAN zoo: `https://deephunt.in/the-gan-zoo-79597dc8c347?gi=4428c735f538`

[7]the *Nash Equilibrium* is the solution of a game, as opposed to a local optimum, which is the solution of an optimization problem

of generating samples which fool the discriminator so that it categorizes them as real samples. When the algorithm converges, the random variables have become into latent variables and when z is provided, the generator generates a sample x drawn from p_{model} (where ideally $p_{model} = p_{data}$). It is easy to empirically check that the latent variables encode, in some way, the knowledge of the generated samples: see figure 2.15 for an example. It is commonly said as an analogy that the generator plays the role of a counterfeiter.

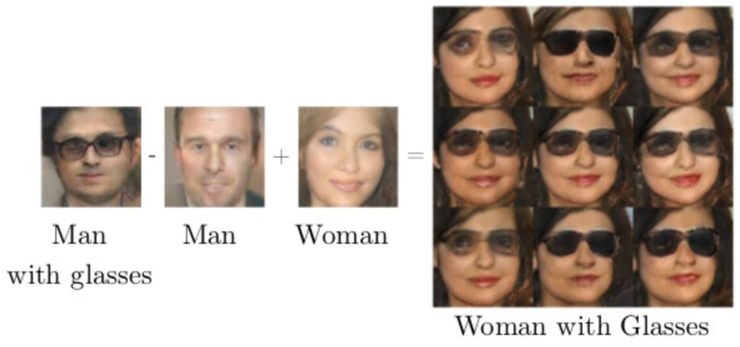


Figure 2.15: Ian Goodfellow illustrated in the *NIPS 2016 conference* tutorial [59] how it was possible to perform arithmetic operations in the input latent vector space. If a vector representing an “average man” is subtracted from a “man with sunglasses” vector and a vector corresponding to an “average woman” is added, then the resulting vector generates an image of a “woman with sunglasses”. The fact that the semantics of the images are preserved in the vector space proves that there is some knowledge encapsulated in the vector space. However, in vanilla GANs there is not a trivial way of treating the features encoded in the vector space because they are not orthogonalized (i.e. the features can be linear combinations of several input variables).

The discriminator takes x as input data and $\theta^{(D)}$ as parameters and plays the role of a policeman which decides if its input samples are real or fake. It consists of a network implementing a binary classifier through traditional supervised learning techniques. The output of the discriminator is the only part of the whole system in which the loss is calculated; afterwards the gradient of the error is backpropagated through all the network.

Assuring derivability in all the components of the whole system (generator and discriminator) is *sine qua non condicio* without which it would not be possible to train the algorithm; the gradient needs to be propagated from the output of the discriminator to the input of the generator.

Both players have a cost function assigned which depends on both parameters sets ($\theta^{(G)}, \theta^{(D)}$) but one player cannot control the other's parameters (see equations 2.10 and 2.11, where $J^{(D)}$ refers to the discriminator cost function and $J^{(G)}$ refers to the generator cost function). That is why it is considered a game and not an optimization problem and that is also where the “adversarial” term comes from.

$$\theta^{(D)} = \arg\min_{\theta^{(D)}} J^{(D)}(\theta^{(D)}, \theta^{(G)}) \tag{2.10}$$

$$\theta^{(G)} = \arg\min_{\theta^{(G)}} J^{(G)}(\theta^{(D)}, \theta^{(G)}) \tag{2.11}$$

Ian Goodfellow presented 2 options with regard to the cost functions. The first one consists of a zero-sum game (also called minimax) in which the discriminator cost $J^{\theta^{(D)}}$ is the average crossentropy loss of two minibatches, one generated and one real (see equation 2.12), and the generator loss is simply $J^{(G)} = -J^{(D)}$. This is an elegant solution because it represents a minimax game but in practice it thrives to convergence issues [8]. To solve them, an alternative loss for the generator is presented in equation 2.13 where, in the case of the generator, only the minibatch of the generated data is involved in its parameters optimization. This reformulation of the problem makes the system not being a minimax game, but it alleviates the problem of the vanishing gradient. In the first formulation the generator tries to fool the discriminator by making it fail in all their predictions, while in the new formulation it tries to fool the discriminator by making it fail in their predictions over the generated samples. The author highly encourages using the second formulation.

$$J^{(D)}(\theta^{(D)}, \theta^{(G)}) = -\frac{1}{2} \cdot E_{x \sim p_{data}} (\log D(x)) - \frac{1}{2} \cdot E_z \log (1 - D(G(x))) \tag{2.12}$$

$$J^{(G)}(\theta^{(D)}, \theta^{(G)}) = - \cdot E_z \log (D(G(x))) \tag{2.13}$$

The training process consists of simultaneous *SGD* updates and is described in algorithm 1 and in the figure 2.16

2.3.4 Comparison between generative model techniques

As a recap, the main advantages and disadvantages of each generative modeling methodology are described in table 2.1.

[8] when the discriminator becomes too strong, the gradient signal of the generator vanishes and the system diverges [59]

Table 2.1: Advantages and disadvantages of the most common generative modeling techniques

	Advantages	Disadvantages
Autoregressive models (FVBNs)	- Provide a fully tractable density function which allows to directly optimize it. - Produce a very good quality estimator. - Proved to work well with continuous and discrete data	- Impossible to parallelize which make the model have linear complexity with the size of the input. - Does not correctly handle multiple solutions problems.
Variational Autoencoders	- Provide an approximation to the density function, allowing sampling from it. - Allow going back and forward (i.e. inferring latent codes from existing samples). - Proved to work well with continuous and discrete data.	- Produce low-quality samples. - The quality estimate is not as good as the FVBN ones. - Does not handle correctly multiple solutions problems.
Generative Adversarial Networks	- Produce very good quality samples. - Naturally handles multiple solutions problems. - Have a higher degree of freedom in terms of architectural. design	- Do not provide any explicit density function. - Difficult to train. - Lack of robust quality estimator (has to be engineered).

Algorithm 1 *GAN* training process as explained in the original paper. Generally, Adam optimizer is chosen (by recommendation of the author), equal learning rates for G and D ($\alpha \leftarrow 0.0001$), and minibatch size of $m = 128$

```
Require: $\alpha$, the learning rate. $(\theta^{(D)}, \theta^{(G)})$, the initialized parameters of $D$
    and $G$ respectively. $data$, a set of observed examples to sample from. $Z$, a
    random distribution to sample from. $m$, the minibatch size.
1: while $(\theta^{(D)}, \theta^{(G)})$ have not converged do
2:     Sample $\{x^{(i)}\}_{i=1}^{m} \sim p_{data}$ a minibatch of real data
3:     Sample $\{z^{(i)}\}_{i=1}^{m} \sim p_{Z}$ a minibatch of prior samples
4:     $g_D \leftarrow \nabla_{\theta^{(D)}} J^{(D)}(x, z, \theta^{(D)}, \theta^{(G)})$
5:     $g_G \leftarrow \nabla_{\theta^{(G)}} J^{(G)}(z, \theta^{(D)}, \theta^{(G)})$
6:     $\theta^{(D)} \leftarrow \theta^{(D)} - \alpha \cdot Adam(\theta^{(D)}, g_D)$
7:     $\theta^{(G)} \leftarrow \theta^{(G)} - \alpha \cdot Adam(\theta^{(G)}, g_G)$
8: end while
```

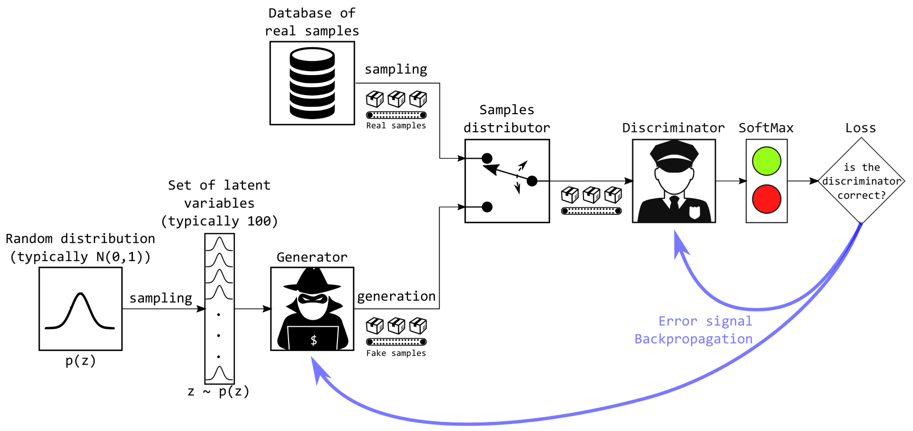


Figure 2.16: *Generative Adversarial Network* general procedure.

Chapter 3

Methods

Despite the great ability of the *Generative Adversarial Networks* (*GAN*) of producing very sharp and visual-appealing images, they have been branded as being remarkably difficult to train [9, 67, 68, 69]. The generator and the discriminator losses must be balanced so that no one of them is much bigger than the other one, otherwise the algorithm diverges. The algorithms described in this section, *WGAN* and its *WGAN-GP* variation, represent successful attempts to solve these issues. *WGAN* is the first approach that successfully stabilizes *GANs* but it still presents some inconsistencies, specially when working with complex architectures. *WGAN-GP* starts from *WGAN* and implements some modifications that solve, or at least effectively reduce, these inconsistencies.

The text generation is a very challenging task for *Generative Adversarial Networks*, as stated in the section 1.3, and that is why *WGAN* and *WGAN-GP* have been chosen as the first candidates for attempting to solve it.

3.1 Wasserstein Generative Adversarial Networks (WGAN)

In 2017 *Arjovsky et al.* presented the *Wasserstein GAN* [9], a variation of the GAN algorithm that solved most of the difficulties that ocurred when training GANs. As proved by [70], the losses of the generator and the discriminator of a GAN must be balanced so that the discriminator does not achieve the optimal point, otherwise either the generator gradients vanish, or the generator updates become very unstable (depending on the formulation that is chosen for the GAN [8]).

As it is broadly discussed in [9], the divergence metrics that the original formulation minimizes, lead to a potentially not-continuous optimization space with respect to the generator parameters, leading to training difficulties. That motivates the author to propose minimizing the *Wasserstein-1* distance (abbreviated as *Wasserstein* distance from now on) which is continuous and differentiable in almost all the space if the whole system is a *Lipschitz* function [9]. The

Wasserstein distance can be informally understood as follows: if the two distributions to be compared (p and q) are viewed as piles of dirt, the *Wasserstein* distance would provide the cost of optimally transporting mass from one distribution to another in order to transform the distribution q into the distribution p. The use of this new distance makes it possible to train the discriminator to optimality, assuring that the further generator updates are accurate.

A function $f : X \to Y$ is K-Lipschitz if there exists a real $K \geq 0$ such that for all x_1 and x_2 in X it satisfies the equation 3.1, where d is a metric function. In a more straightforward way, a function is K-Lipschitz if its derivatives are bounded to the $[-K, K]$ range.

$$d(f(x_1), f(x_2)) \leq Kd(x_1, x_2) \tag{3.1}$$

The cost function of the *WGAN* is defined in the equation 3.2. As it can be noticed, by comparison with equations 2.12 and 2.13, the logarithms have been removed. In this new formulation, the discriminator can provide unbounded values, and that is why it is so-called *critic* instead of discriminator (think of an art critic as an analogy). This notation will be adopted from now on.

$$\min_G \max_D E_{x\sim P_r}[D(x)] - E_{\tilde{x}\sim P_g}[D(\tilde{x})] \tag{3.2}$$

In order to assure *Lipschitz* continuity, the authors propose to clip the weights of the critic network to c, i.e. after each critic update, all the weights of the network belonging to the critic are trimmed to belong the $[-c, c]$ range. The author recognizes that though this procedure assures *Lipschitz* continuity, it is too rough and needs for further investigation, but it still yields to good performance, improving the stability and results of the original formulation. The c parameter must be correctly chosen; if it is too large the critic needs too updates to reach optimality, if it is too small vanishing gradient issues can appear [9]. The whole training process is described in algorithm 2.

Empirical trials of this method shown improved stability of the optimization process. Mode collapse[1] [9] is also reduced when compared with traditional GAN systems.

A comparison of the generated samples between GAN and WGAN is shown in figure 3.1.

3.2 Wasserstein Generative Adversarial Networks with gradient penalty (WGAN-GP)

Gulrajani et al. presented in the *NIPS 2017* conference a modification of the original *WGAN* [71]. They basically replaced the weight clipping by a penalty term of the critic gradient norm with respect to its input. The authors start

[1]it is a failure mode that is frequently observed in some GAN architectures. When it occurs, the generator produces always almost the same image (or set of images); i.e. the variability of the generated samples is dramatically reduced.

Algorithm 2 *WGAN* training process as explained in the original paper.The default values provided by the author are: *RMSprop* optimizer, $\alpha \leftarrow 0.00005$, $c \leftarrow 0.01$, $n_{critic} = 5$ and minibatch size of $m = 64$

```
Require: $\alpha$, the learning rate. $(\theta^{(D)}, \theta^{(G)})$, the initialized parameters of $D$ (the
    critic) and $G$ (the generator) respectively. $data$, a set of observed examples
    to sample from. $Z$, a random distribution to sample from. $m$, the minibatch
    size. $c$, the clipping parameter. $n_{critic}$ the number of iterations of the critic
    per generator iteration.
 1: while $\theta^{(G)}$ has not converged do
 2:     for $t = 0, ..., n_{critic}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^{m} \sim p_{data}$ a minibatch of real data
 4:         Sample $\{z^{(i)}\}_{i=1}^{m} \sim p_Z$ a minibatch of prior samples
 5:         $g_D \leftarrow \nabla_{\theta^{(D)}}[\frac{1}{m}\sum_{i=1}^{m} f_{\theta^{(D)}}(x^{(i)}) - \frac{1}{m}\sum_{i=1}^{m} f_{\theta^{(D)}}(g_{\theta^{(g)}}(z^{(i)}))]$
 6:         $\theta^{(D)} \leftarrow \theta^{(D)} - \alpha \cdot RMSProp(\theta^{(D)}, g_D)$
 7:         $\theta^{(D)} \leftarrow clip(\theta^{(D)}, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^{m} \sim p_Z$ a minibatch of prior samples
10:     $g_G \leftarrow -\nabla_{\theta^{(G)}} \frac{1}{m}\sum_{i=1}^{m} f_{\theta^{(D)}}(g_{\theta^{(g)}}(z^{(i)}))$
11:     $\theta^{(G)} \leftarrow \theta^{(G)} - \alpha \cdot RMSProp(\theta^{(G)}, g_G)$
12: end while
```

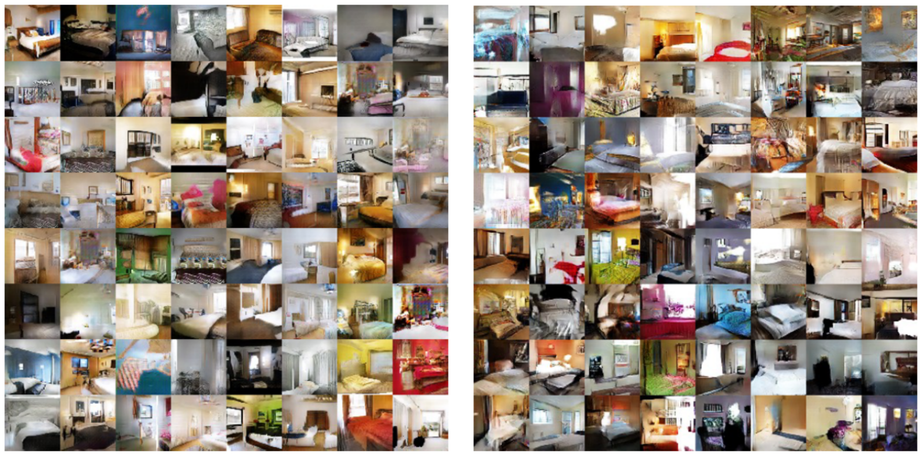


Figure 3.1: Comparison between *GAN* and *WGAN* using the *LSUN* image dataset. As it can be seen the results are quite similar, but the *WGAN* training process is much more stable than the *GAN*, allowing the algorithm to work with more complex architectures.

providing a set of arguments that prove the difficulties introduced by applying weight constraints in the *WGAN* algorithm; they are summarized below.

- The critic function achieved when applying weight constraints is biased towards much simpler functions, ignoring higher moments in the data distribution. That capacity under-use in the critic yields to very poor approximations of the real data distribution in the generator (see figure 3.3-left).
- When the weight clipping threshold c is not carefully chosen it results in either vanishing gradients or exploding gradients (see figure 3.3-right).

These issues and the experimental results motivate the authors of [71] propose an alternative way of enforcing the Lipschitz continuity constraint: penalizing the gradient norm of the critic with respect to its inputs. In order to make it tractable, [71] propose using a softer version of the constraint by introducing random samples $\hat{x} \sim \mathbb{P}_{\hat{x}}$. The *WGAN-GP* critic objective, defined in equation 3.3 is then composed of two components: the original *WGAN* critic loss defined by [9] plus the gradient norm penalty introduced by [71].

$$L_{critic} = \underbrace{\underset{\tilde{x}\sim\mathbb{P}_g}{\mathbb{E}}[D(\tilde{x})] - \underset{x\sim\mathbb{P}_r}{\mathbb{E}}[D(x)]}_{\text{original WGAN critic loss}} + \underbrace{\lambda \underset{\hat{x}\sim\mathbb{P}_{\hat{x}}}{\mathbb{E}}[(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1)^2]}_{\text{gradient norm penalty}} \tag{3.3}$$

The sampling distribution given by $\hat{x} \sim \mathbb{P}_{\hat{x}}$ is defined as sampling uniformly along straight lines between the real data distribution $\mathbb{P}_r$ and the implicit generator distribution $\mathbb{P}_g$. This is proposed as an alternative method for locally enforcing the *Lipschitz* constraint, because enforcing it everywhere is intractable [71]. Gulrajani et al. argue that this approach yields good performance models in practice. Regarding the penalty coefficient (λ), the authors recommend to fix it to $\lambda = 10$ as a rule of thumb, because it showed good results in a wide variety of tasks and architectures. The whole algorithm procedure is described in Algorithm 3.

The *WGAN-GP* loss is not compatible with the use of batch normalization in the critic architecture because the gradient penalty is applied at input level, not at batch level[2]. The authors propose using layer-normalization, instead.

A comparison a set of generated samples between *GAN* and *WGAN* is shown in figure 3.2.

[2]applying batch normalization using a batch size of 1 is also not an option because empirical trials that will be described in the results section showed a strong dependency of the quality of the generations with the batch size

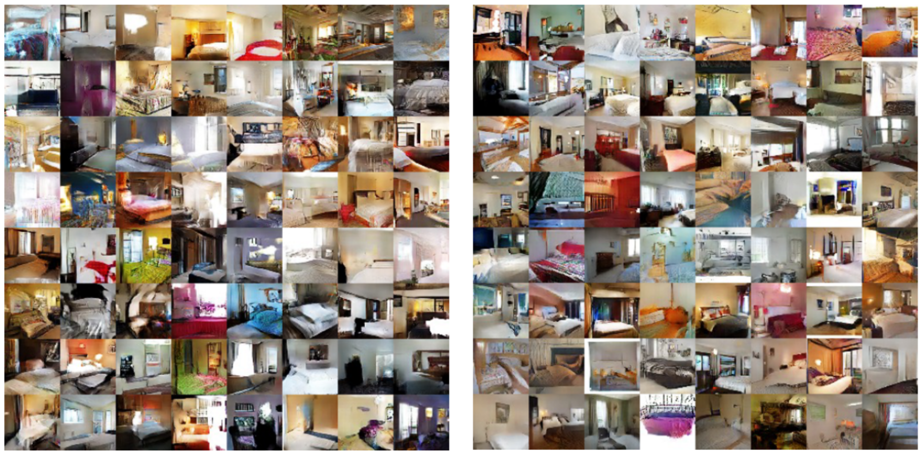


Figure 3.2: Comparison between *WGAN* and *WGAN-GP* using the *LSUN* image dataset. As it can be seen the results are very similar, but the *WGAN-GP* modification produces a much more stable model, which in practice converges more easily. This version also allows the algorithm to converge with more complex architectures.

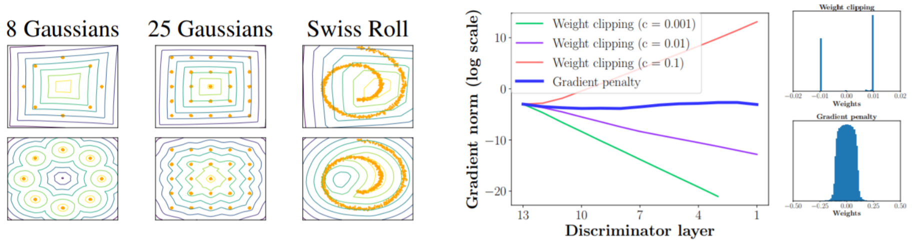


Figure 3.3: Representation of the issues happening in *WGAN* (with weight clipping) in comparison with the results obtained with *WGAN-GP* (*WGAN* with gradient penalty). Figure extracted from the *Gulrajani et al.* original paper [71]. The figure in the left shows the value surfaces for the critics trained optimally on toy datasets, using the original *WGAN* formulation (top), and the *WGAN-GP* formulation (bottom). It is easy to see how the original *WGAN* ends up producing extremely simple critic value functions while the *WGAN-GP* achieve much more complex ones. The figure in the right shows how original *WGAN* produces either vanishing or exploding gradients depending on the value of c while the new *WGAN-GP* achieves much more stable gradients.

Algorithm 3 WGAN-GP training process as explained in the original paper.The default values provided by the author are: *Adam* optimizer with $\alpha \leftarrow 0.0001$, $\beta_1 = 0$, $\beta_2 = 0.9$; $n_{critic} = 5$, $\lambda = 10$ and minibatch size of $m = 64$

```
Require: $\alpha$, the learning rate. $(\theta^{(D)}, \theta^{(G)})$, the initialized parameters of $D$ (the
    critic) and $G$ (the generator) respectively. $data$, a set of observed examples
    to sample from. $Z$, a random distribution to sample from. $m$, the minibatch
    size. $\lambda$, the gradient penalty parameter. $n_{critic}$ the number of iterations of
    the critic per generator iteration.
 1: while $\theta^{(G)}$ has not converged do
 2:     for $t = 0, ..., n_{critic}$ do
 3:         for $i = 0, ..., m$ do
 4:             Sample $\epsilon \sim U(0,1)$ a random number between 0 and 1
 5:             Sample $x \sim p_{data}$ real data
 6:             Sample $z \sim p_Z$ a prior sample
 7:             $\tilde{x} \leftarrow G_{\theta^{(G)}}(z)$
 8:             $\hat{x} \leftarrow \epsilon x + (1-\epsilon)\tilde{x}$
 9:             $L^{(i)}_{(D)} \leftarrow D_{\theta^{(D)}}(\tilde{x}) - D_{\theta^{(D)}}(x) + \lambda(\|\nabla_{\hat{x}} D_{\theta^{(D)}}(\hat{x})\|_2 - 1)^2$
10:         end for
11:         $\theta^{(D)} \leftarrow \text{Adam}(\nabla_{\theta^{(D)}} \frac{1}{m} \sum_{i=1}^{m} L^{(i)}_{(D)}, \theta^{(D)}, \alpha, \beta_1, \beta_2)$
12:     end for
13:     Sample $\{z^{(i)}\}_{i=1}^{m} \sim p_Z$ a minibatch of prior samples
14:     $\theta^{(G)} \leftarrow \text{Adam}(\nabla_{\theta^{(G)}} \frac{1}{m} \sum_{i=1}^{m} -D_{\theta^{(D)}}(G_{\theta^{(G)}}(z)), \theta^{(G)}, \alpha, \beta_1, \beta_2)$
15: end while
```

Chapter 4

Proposal

The neural network architecture design represented a considerably big part of the effort done in this work. The process of coming with a good architecture took up to 37 different architecture iterations. Many of them didn't work well while some others were promising at the beginning and had to be tested for a long time to see if they worked better than the previous ones.

The two big families of neural networks have been used in order to be as exhaustive as possible in the attempt of generating text with *Generative Adversarial Networks*: *Convolutional Neural Networks* and *Recurrent Neural Networks*. Both types of neural networks have been combined in 37 different ways, varying their principal parameters and two of them (the ones which performed better), are described in detail in this section.

The first idea was building a network composed of a *recurrent neural network* in the generator and a *recurrent neural network* in the critic. In the implementation phase a technical difficulty was found when building a critic using *recurrent neural networks* while applying the gradient penalty term: neither *tensorflow* nor *pytorch* support second order derivatives applied to *recurrent neural networks*. That difficulty has been broadly discussed over the *tensorflow* and *pytorch* community and there is not an easy way of overcome it; that is the reason why this solution was discarded.

Among the 37 architecture iterations, there were two of them that achieved significantly better results than the other ones. The *convolutional-convolutional* and the *RNN_decoder-convolutional* architectures; they will be called *CNN* architecture and *RNN* architecture (respectively) for brevity. The results of the next section will be focused on these two settings.

Details about the 37 architectures that were tested and their main features are summarized in appendix A. For further details, the *GitHub* repository[1] of the project can be reviewed. In chapter 6, the main take aways of the experimental process have been summarized. Each of the architecture iterations can be found in the repository. They have been arranged in different branches. Appendix A

[1] `https://github.com/ivallesp/scriptGAN`

contains a table summarizing further details of each of the versions in *GitHub*.

4.1 RNN architecture

The *RNN* architecture is composed of a *recurrent neural network* setting in the generator and a *convolutional neural network* in the critic. The details of the architecture are described below and summarized in the figure 4.1.

The generator has three blocks: (1) a projection block which adapts the latent vector of the input to the initial state of the recurrent block, (2) the recurrent block formed of *LSTM* (*Long-Short Term Memory*) cells of size 1024, arranged in a single layer and (3) a block of dense layers applied over each sequence step separately that is intended to adapt the output of the generator to the required shape before applying the *softmax* transformation.

The projection block consists of 2 dense layers that take the latent vector z (with d_z elements) and produce two vectors of size 1024 (the same size as number of units in the *LSTM*). These two vectors will be the initial states (h and c) of the *recurrent module*.

The *LSTM* cells of the recurrent block are arranged in a decoder structure [72], which has the following features.

- It receives a context in form of two vectors as initial state which, in this case, is a projection of the random latent vector z.
- The outputs of the $t-1$ sequence step are forwarded to the inputs of the t sequence step without any treatment.
- The first input is a trigger signal (also known as the `<GO>` symbol), which in this case is as simple as a vector of ones. Intuitively it is a way of telling the network that this sequence step will be the start of a new sentence.

As stated in appendix A, a *multi-layer RNN* setting has been tested producing worse results.

All the generated sequence steps are passed to the block of dense layers to refine the predictions. In this block, the output of the recurrent block is reshaped so that the timesteps of the sequences are processed independently, that was intended to help the model producing *one hot encodding* similar outputs. At the end of the *dense layers*, the output is reshaped again to its original set of 3-dimensions and a *softmax* is applied to all the sequence steps in order to normalize the outputs.

Batch Normalization has been applied over all the generator architecture producing, in practice, a general improvement of the stability of the system. That happens because the *batch normalization* operation over the layers of a neural network reduces the covariance shift [73]. Figure 4.1a shows the generator architecture.

The critic is built up using *convolutional* and *average pooling* layers due to the second-order derivative constraints imposed by *TensorFlow*. It is based on

4 *convolutional blocks* consisting of: 2 *1D-convolutions* of size 9 and stride 1 and an average pooling layer just afterwards with a size of 2 and a stride of 2. The size of the last average pooling layer is increased to 8 in order to reduce all the sequence to a single number. This number represents the output of the critic. The complete critic architecture is shown in figure 4.1b. The size of the convolution has been fixed to 9 because it showed both, stability and high performance. Larger convolution sizes tend to turn the *GAN* more unstable while smaller sizes produce worse performance. The average pooling size and stride has been fixed to 2 in order to reduce the size of the sequence by half after each *convolutional block*.

There are some *meta-parameters* (described below) that represent design decisions and that must to be also taken into consideration.

- The batch size used to train this network has been $B = 512$ and the size of the latent vector z has been fixed to $d_z = 100$. A strong relation between the batch size and the model performance has been found: the bigger the batch, the better the performance.
- *Adam* has been chosen as optimizer for the generator and the critic, with learning rates of $1 \cdot 10^{-5}$ and $1.5 \cdot 10^{-5}$, respectively. Higher learning rates empirically shown more chances to destabilize the model.
- The learning rate of the critic has been chosen slightly larger than the one of the generator following the recommendations on training the critic until optimality before each generator step (proposed by [9] and [71]).
- The number of successive adjustments of the critic has been fixed to $n_{critic} = 10$, which showed good results in practice in all the trials.

4.2 CNN architecture

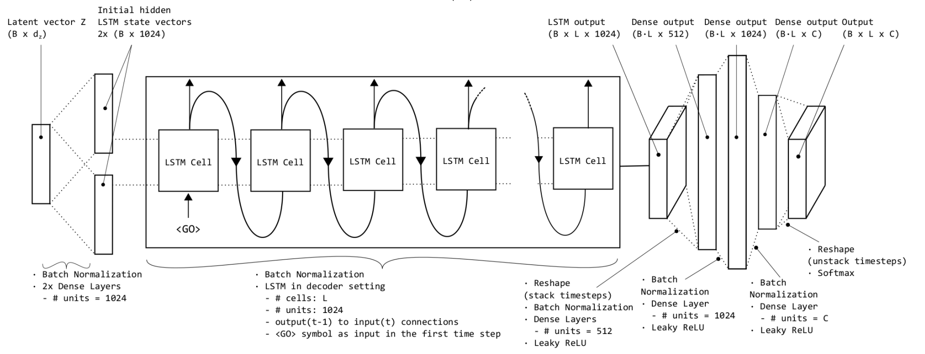
The *CNN* architecture uses the same critic as the *RNN* architecture described in section 4.1. The only difference of this architecture over the *RNN* one is the generator architecture which, in this case, uses a *convolutional neural network*.

The *convolutional neural network* in the generator is quite simple. It consists of three blocks: (1) projection of the z vector, (2) 4 *1d-convolutional* blocks to transform the input to sequences of text, and (3) a final *1d-convolution* that produces C values per character.

The meta-parameters used in the *RNN* architecture (described in section 4.1) remain the same in this version.

Figures 4.2a 4.2b show the generator and critic architectures in form of a schema.

(a)



(b)

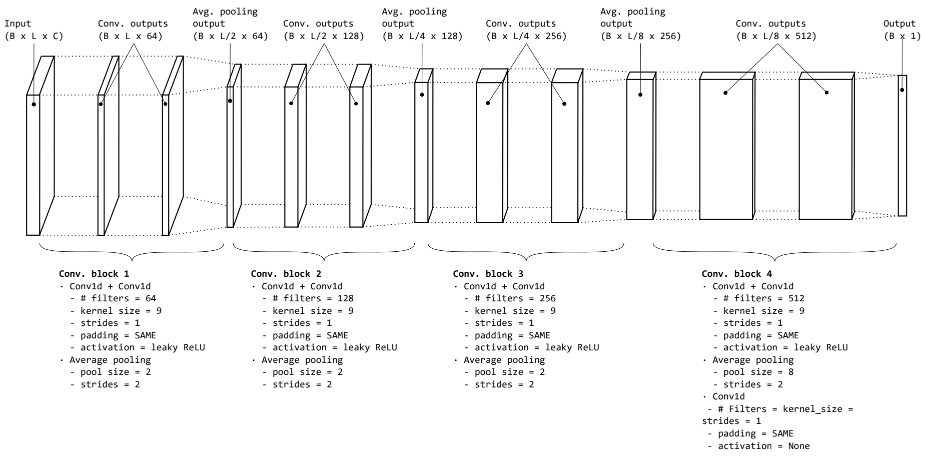


Figure 4.1: Recurrent architecture. (a) generator architecture, with a projection module for adapting the latent vector to the initial state of the recurrent module, a recurrent module with 1024 *LSTM* units and L timesteps in a *decoder* setting, and a set of dense layers for adapting the output before the *softmax* transformation. (b) critic architecture, only containing a combination of *convolutional* layers and *average pooling* so that each input sentence, of length L and C variables, is progressively reduced to a single number.

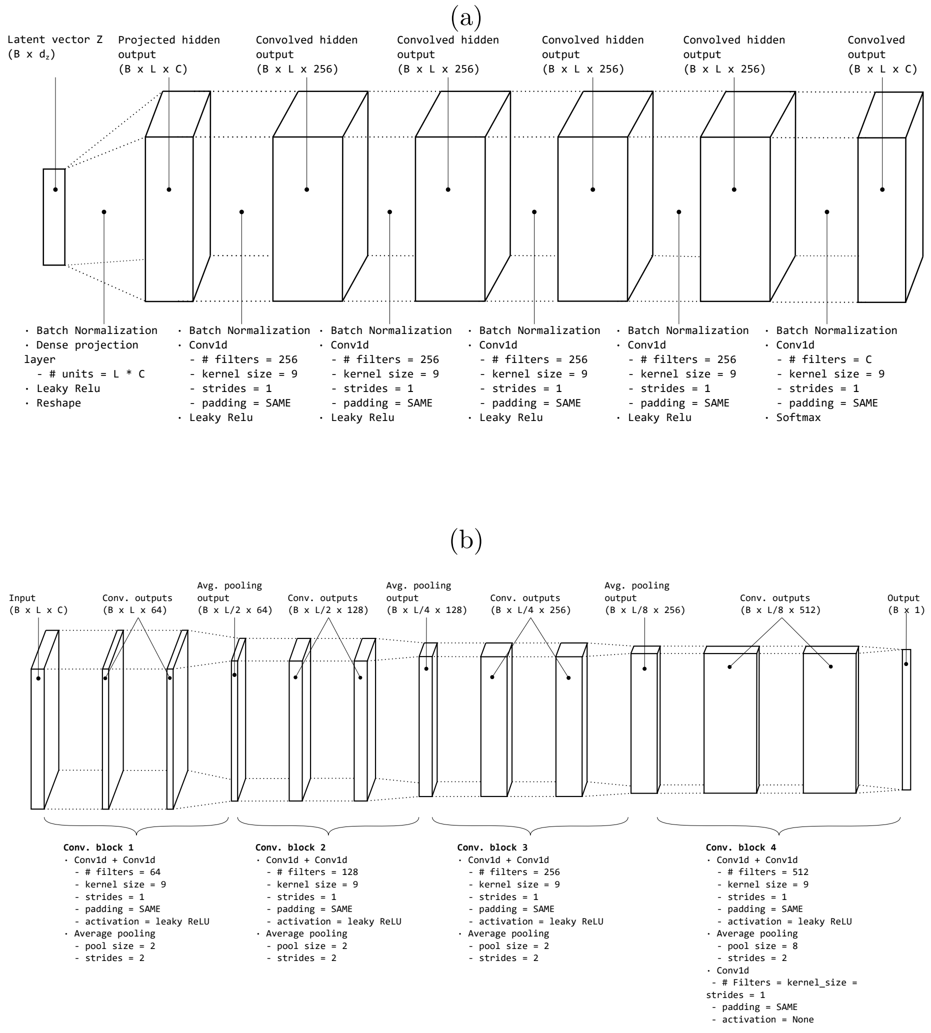

(a)
Latent vector Z (B x d_z)
Projected hidden output (B x L x C)
Convolved hidden output (B x L x 256)
Convolved hidden output (B x L x 256)
Convolved hidden output (B x L x 256)
Convolved hidden output (B x L x 256)
Convolved output (B x L x C)
· Batch Normalization · Dense projection layer - # units = L * C · Leaky Relu · Reshape
· Batch Normalization · Conv1d - # filters = 256 - kernel size = 9 - strides = 1 - padding = SAME · Leaky Relu
· Batch Normalization · Conv1d - # filters = 256 - kernel size = 9 - strides = 1 - padding = SAME · Leaky Relu
· Batch Normalization · Conv1d - # filters = 256 - kernel size = 9 - strides = 1 - padding = SAME · Leaky Relu
· Batch Normalization · Conv1d - # filters = 256 - kernel size = 9 - strides = 1 - padding = SAME · Leaky Relu
· Batch Normalization · Conv1d - # filters = C - kernel size = 9 - strides = 1 - padding = SAME · Softmax
(b)
Input (B x L x C)
Conv. outputs (B x L x 64)
Avg. pooling output (B x L/2 x 64)
Conv. outputs (B x L/2 x 128)
Avg. pooling output (B x L/4 x 128)
Conv. outputs (B x L/4 x 256)
Avg. pooling output (B x L/8 x 256)
Conv. outputs (B x L/8 x 512)
Output (B x 1)
Conv. block 1 · Conv1d + Conv1d - # filters = 64 - kernel size = 9 - strides = 1 - padding = SAME - activation = leaky ReLU · Average pooling - pool size = 2 - strides = 2
Conv. block 2 · Conv1d + Conv1d - # filters = 128 - kernel size = 9 - strides = 1 - padding = SAME - activation = leaky ReLU · Average pooling - pool size = 2 - strides = 2
Conv. block 3 · Conv1d + Conv1d - # filters = 256 - kernel size = 9 - strides = 1 - padding = SAME - activation = leaky ReLU · Average pooling - pool size = 2 - strides = 2
Conv. block 4 · Conv1d + Conv1d - # filters = 512 - kernel size = 9 - strides = 1 - padding = SAME - activation = leaky ReLU · Average pooling - pool size = 8 - strides = 2 · Conv1d - # Filters = kernel_size = strides = 1 - padding = SAME - activation = None

Figure 4.2: CNN architecture. (a) generator architecture containing only *convolutional* layers after the projection step. All the *convolutions* in this architectures are the same size and all of them (but the last one) are based on 256 filters. The last one has the same number of filters as the size of the character set (i.e. C). (b) critic architecture, only containing a combination of *convolutional* layers and average pooling so that each input sentence, of length L and C variables, is progressively reduced to a single number.

Chapter 5

Experimental setting and results

5.1 Data

In this study, the distribution of a *dataset* is intended to be learned. The chosen *dataset* must have a set of features in order to be eligible for this study; they are briefly described below.

- It has to be formed of sentences because the algorithm is intended to write text at sentence level.

- The sentences in the real dataset must be *morphologically*, *syntactically* and *semantically* correct. That will be achieved only if the quality of the data is good enough. The reason why it is required is because the algorithm is going to learn from the distribution of data which is provided; if a bad quality data is provided, the generations will be bad quality too.

- The data must have high variability so that the model can learn a spread distribution, otherwise the model will be prone to mode collapse (getting stuck in modes of the distribution, i.e. generating almost the same sentence every time).

- The data must represent general usage of the language (English in this case) in order to draw generalizable conclusions. The reason why no different trials with different *datasets* have been conducted is because the models applied in this work are very computational expensive and running the algorithms through different *datasets* is unaffordable.

- The sentences in the dataset must not be very long so that the memory required by the data lets the complexity of the architecture be big enough to model the language successfully.

One of the biggest existing corpus containing bilions of sentences that can be found on the internet is *twitter*. That is the reason why in the begining of this work *twitter* data were considered as the main data distribution to learn from. The main benefit of using this *corpora* is that the sentences contained on it (the *tweets*) are constrained to a maximum length. Nevertheless, due to the complexity of the modeling task, *twitter* data turned out to be too noisy because of the following reasons.

- *Twitter* uses *hashtags* and *mentions*, which are quite dependent on the temporal context and on the topology of the social network. This information is very difficult to model by an algorithm that only sees at the *twitter* streamline without any order and does not consider the surrounding context and the users relations beyond the social network.
- Most of the *tweets* contain plenty of acronyms, abbreviations and informal language which significantly complicates the data distribution.
- Insolating the *tweets* belonging to a specific language is not a straightforward task, specially with such an informal register.
- It is difficult to get content with high variability because the users and the *hashtags*, by definition, are biased. The majority of the *twitter* publicly available data are collected by selecting a random sample of users or one or a set of *hashtags*, instead of selecting a real random sample of tweets.

For that reasons, a more curated *corpora* was desired. After an exhaustive research *Tatoeba corpus*[1] was found. *Tatoeba* is a large open-source and free collection of sentences written in multiple languages which is intended to be a powerful resource for natural language processing tasks. It is specially focused on machine translation tasks. The sentences contained in this *corpus* have been included by the community of registered users and they have an impressive quality. A sample of english sentences extracted from this collection is included below as an example.

```
Is it cruel to declaw your cat?
I learned English words by heart all day yesterday.
I knew that someone would come.
Any gentleman would not use such language.
Mary is still living with her parents.
The eastern sky was getting light.
You are weak.
I very seldom eat lobster.
Those are the leftovers from lunch.
Do you like your new apartment?
It took him three tries.
Tom walked into his room.
```

[1]https://tatoeba.org/eng/

```
I'd like to thank you all for coming today.
Tom was elected captain.
I'll throttle him!
Tom really inspired me.
Tom didn't understand Mary's joke.
I want to send this letter to Japan.
He's wearing sunglasses.
Tom will unlikely have to do that anytime soon.
With Fadil, Layla finally found the excitement she had been craving.
Don't talk about my daughters like that.
Can we help?
Mary is saving money so she can go to Japan.
They weren't busy.
He put his arm around my waist.
Didn't you know that the east Asian New Year is today?
Rarely have I heard such a load of rubbish.
She came with her hands in her pockets.
You like to sing, don't you?
Tom claims he was drunk at the time.
I'm glad you accepted my offer.
Mary says she has to speak French.
Don't attempt to do this by yourself.
My shift's over.
It's going to be easy to do that.
Are there many Chinese restaurants in Boston?
Tom didn't seem surprised when I told him I didn't need to do that.
```

The sentences in this collection have been slightly preprocessed in order to have an even cleaner *corpus*. The steps followed to get the clean set are summarized below. In addition, figure 5.1 shows the quantity of filtered sentences in each step.

- The sentences labeled as belonging to a language different from english were removed (the collection was reduced from 6.2 million sentences to 890.2 thousand).
- The sentences longer than 64 characters were discarded (the colection was reduced to 829.8 thousand).
- The sentences containing characters which do not belong to the set of the 100 most common characters across all the english sentences were removed (the collection was reduced to 828.9 thousand)

At the end of the preprocessing phase, two synthetic symbols were appended to each sentence: a `<START>` symbol at the begining and an `<END>` symbol at the end. The sentences that are shorter than $L = 64+2$ are right padded with a third synthetic symbol `<UNK>`. These preprocessing steps are necessary so that

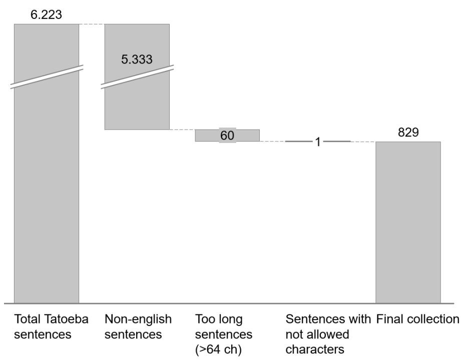


Figure 5.1: Filters applied over the original *Tatoeba* collection and number of sentences (in thousands) affected.

all the sentences contain the same number of characters in order to be stacked together in a 3-dimensional matrix to form the *minibatches*. The characters of the sentences in this stage are converted to a 103-elements binary vectors using *one hot encoding*. These vectors have been stacked together so that each sentence forms a matrix of shape (L x C), where $L = 66$ is the maximum length of the sentences and $C = 103$ is the cardinality of the character set.

5.2 Metrics and KPIs

One of the most discussed difficulties when dealing with *Generative Adversarial Networks* is the lack of a robust evaluation method, then there is not such an automatic way of quantifying how good the generated samples are. In the last years there has been a substantial amount of research in this area and some techniques have been developed for measuring how appealing some generated images are to the human judgment [74, 75, 76]. This is not the case of the text, probably because the vast majority of the research studies on *GANs* are related to images.

In order to compare between architectures, an evaluation metric is needed. *BLEU* score [77] is a widely used metric to automatically evaluate machine translation tasks, but in this case, as there are not specific target sentences to compare with, it is not possible to use it directly. Instead, a more simple approach has been proposed: using the *n-gram precision* scores described in [77]. As stated by the authors, they are not robust metrics, but still provide with a way of comparing between models and are strongly correlated to the human judgement. These metrics are described below. In the following definitions, let R be the distribution of real samples and $|S|$ the number of words in the sentence. All the following equations are defined for a given sentence S drawn from G, the distribution of generated samples.

1-gram precision

Measures the percentage of the words in the generated sentences which also appeared at least once in the real dataset (the *Tatoeba* dataset). Equation 5.1 shows how to calculate it, where w represents a word in the generated sentence.

$$Prec_1 = \frac{\sum\limits_{w \in S} g(w)}{|S|} \quad \text{where} \quad g(w) = \begin{cases} 1 & \text{if} \quad w \in R \\ 0 & \text{if} \quad w \notin R \end{cases} \tag{5.1}$$

2-gram precision

Measures the percentage of the pairs of contiguous words in the generated sentences which also appeared at least once in the real dataset (the *Tatoeba* dataset). Equation 5.2 shows how to calculate it where (w_1, w_2) represents a *bigram* (i.e. a combination of two contiguous words appearing in the text) in the generated sentence.

$$\begin{aligned} Prec_2 &= \frac{\sum\limits_{w_1, w_2 \in S} g((w_1, w_2))}{|S| - 1} \\ \text{where} \quad g((w_1, w_2)) &= \begin{cases} 1 & \text{if} \quad (w_1, w_2) \in R \\ 0 & \text{if} \quad (w_1, w_2) \notin R \end{cases} \end{aligned} \tag{5.2}$$

3-gram precision

Measures the percentage of the triplets of contiguous words in the generated sentences which also appeared at least once in the real dataset (the *Tatoeba* dataset). Equation 5.2 shows how to calculate it, where (w_1, w_2, w_3) represents a *trigram* (i.e. a combination of three contiguous words appearing in the text) in the generated sentence.

$$\begin{aligned} Prec_3 &= \frac{\sum\limits_{w_1, w_2, w_3 \in S} g((w_1, w_2, w_3))}{|S| - 2} \\ \text{where} \quad g((w_1, w_2, w_3)) &= \begin{cases} 1 & \text{if} \quad (w_1, w_2, w_3) \in R \\ 0 & \text{if} \quad (w_1, w_2, w_3) \notin R \end{cases} \end{aligned} \tag{5.3}$$

All the defined metrics range from 0 to 1, where 0 means a bad generation and 1 a perfect one. Below, a set of drawbacks related with the use of these metrics are enumerated.

- Longer sentences are more likely to contain mistakes than shorter ones: each of the metrics defined above does not depend on the length of the sentences, so it is easy to assume that they will easily produce higher scores for shorter sentences. For example, the following sentence would score $Prec_1 = Prec_2 = Prec_3 = 100\%$ only containing 9 characters: “`a bad cat`”.

- The real dataset is not perfect: the metrics defined above are based on observations over the original dataset and assume that it does not have errors. For example, if a wrong triplet of words appear in the real dataset, it will always be considered as a valid language construction for measuring the results.

- The real dataset does not fully cover the whole space of possibilities: if the model comes up with a new construction that never appeared in the real dataset, it will be labeled as an error.

- The correctness of a sentence depends on its whole content: the metrics defined above only take into account contiguous triplets of words, which may result in wrong measurements in some special cases. For example, the following sentence would be scored as $Prec_1 = Prec_2 = Prec_3 = 100\%$: “`In the house of the king of the king of the king of the king`”.

- The variability of the generator is not asessed by these metrics. That makes it necessary to manually-review the generated sentences so that the typical *failure modes* of the *GANs*, like mode-collapse, do not happen.

Despite the weaknesses of these metrics, there are some reasons why they have been used in this work; they are summarized below.

- The models proposed in this work are character-based and the metrics proposed are word-based. If no correct words are generated, it is very unlikely that the weaknesses discussed above arise. There is a low probability of getting incorrect good measurements due to this reason

- The metrics defined above have not been used for optimizing the parameters of the models and the design decisions that have been taken are not based on a single *minibatch* measurement.

- Not only these metrics have been evaluated for inferring decisions in the design process, but also manual evaluations over the generated samples have been conducted.

- The *Tatoeba* dataset is quite large and the variability that it covers is also large. That makes the metrics be more robust because a large proportion of the total set of valid constructions appears in the original data.

5.3 Results

In this section the results for the two architectures described in section 4 are both qualitatively and quantitatively discussed. The experiments have been conducted using the *Google TensorFlow 1.3.0* library [78] in *Python 3.6.3* from the *Anaconda* distribution. The results have been logged using *Tensorboard*. The algorithms have been run in a single GPU (*Nvidia Titan XP*) during 65 days to achieve the results presented in this section.

The models have been trained from scratch, without using any pretrained model nor any kind *a priori* knowledge about the language. With the aim of evaluating them, two sets of 5000 sentences have been generated using each of the algorithms.

The achieved results in terms of *n-gram precision* are summarized in table 5.1. It shows the metrics values achieved by both algorithms in their best iteration[2]. The evolution of the metrics in each iteration is shown in figure 5.2. As it can be seen, the *CNN* architecture achieves substantially better results than the *RNN* one. Looking at the tendency in the figures representing the evolution of the metrics, it seems that the *RNN* architecture has converged to its optimal point while the *CNN* one seems to be still improving. Both algorithms have been stopped due to a time constraint. All the results and generated samples appearing in this section have been calculated in an out of sample trend, i.e. they have been generated using new randomized z vectors, assuring they are different than the vectors for which the models have been adjusted at training time.

Architecture	Best Iteration	$Prec_1$ $(\mu \pm \sigma)$	$Prec_2$ $(\mu \pm \sigma)$	$Prec_3$ $(\mu \pm \sigma)$
RNN (v14)	1064600	0.538 ± 0.242	0.321 ± 0.239	0.124 ± 0.188
***CNN* (v20)**	**2675000**	$\mathbf{0.673 \pm 0.231}$	$\mathbf{0.457 \pm 0.254}$	$\mathbf{0.186 \pm 0.219}$

Table 5.1: Results for the best iterations of the *RNN* and *CNN* architectures. They have been measured over a sample of 5000 generated sentences using the iteration step of the models which better results showed in the training phase. In bold, the architecture which showed better performance (the *CNN* architecture). For the best case the results can be read as: on average, the 67.3% of the words, the 45.7% of the bigrams and the 18.6% of the trigrams generated by the algorithm are known, meaning that they appeared at least once in the real dataset (*Tatoeba*).

As it has been discussed in section 3, it is very important to manually revise the sentences generated by the algorithms so that most of the complex structure of the language which is not correctly assessed by the metrics, can be assessed by the human judgement. One of the most important things to evaluate is if there is *mode collapse*: in this context, it would be the case if there were almost no variability between the generated sentences; i.e. if they looked like

[2]An iteration consists of one set of adjustments to the discriminator and to the generator

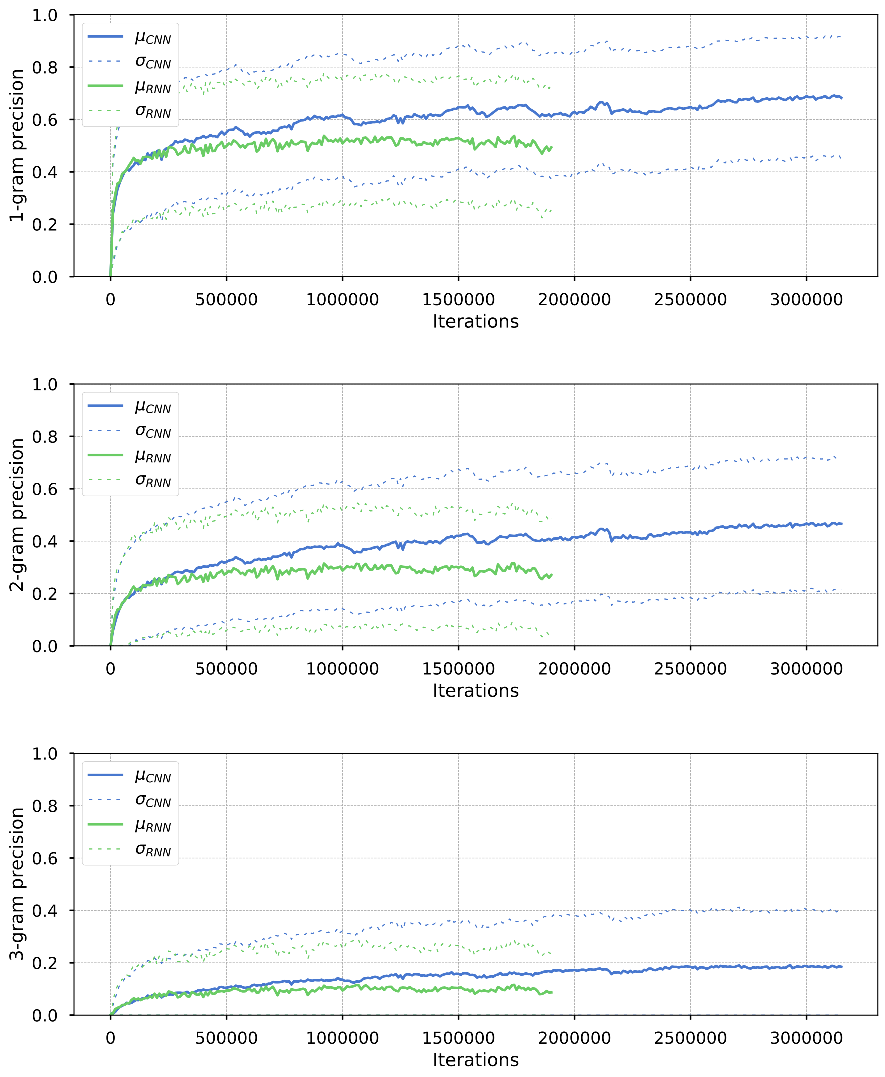


Figure 5.2: Evolution of the three metrics in each iteration step for the *RNN* and the *CNN* algorithms.

the same sentence with small or even no variation. The sentences shown below correspond to the ones which have higher *compound n-gram precision*, calculated as the product of the three metrics ($Prec_1 \cdot Prec_2 \cdot Prec_3$)

Below, a set of the top 40 sentences generated by the *RNN* architecture have been included in order to perform the manual assessment.

```
This is your had to dying all of anymore.
I think Mary will fell hot what Tom going to that much bepach.
I'm noting that happened.
I busy was on the change I long mamied today?
Tom didn't for Tom.
We're minuted at home down.
You are so bick.
Mary old Mary that Tom will how to do that.
I need a treak that you.
I will bigies very your aroud.
It is quile exacely from are even sest a policy.
That's very talking my make beltrietime.
I'm taking tn gee to watched the house.
They were plans to mead, pofice.
I'm splak is my colfer off sere.
Tom know that me.
Can he wae now.
The expender is finitiker, Tom.
Tom is thy very cands.
I was make hom so sugged better the still with you.
I don't think Mary want to help Tom.
Do you want to go alk that where tvey stuld like.
What man wart's a under tone?
Momy aky, you're not adrired.
I'm sure you think Tom knowmed in you and happed her.
I always enterbering up to she on the ous it?
Tom said that you'vera fould here.
I think hars even that Mary were knars.
Tom sumgeng all three book simewhere.
I like to help you?
I just know inythe balle always tho frout.
My prace himing anstous of ealiglans of herself.
You mid this prevent suroubly.
The money pointal.
Tom silled whe longer us again.
The maiturg you manes on flece.
Tom isn't his rest.
Tom forlld doink with this per.
I haven't give met.
We can't like this oves for famich.
```

A set of the top 40 sentences generated by the *CNN* architecture are shown below.

```
Tom and Mary wasn't to do you do here.
Are you going to take.
Tom needed to do?
Tom and Mary doesn't do that.
We have to leave.
Mary should have to want to win?
Tom doesn't know.
I'd like to.
I know what you said you want.
Are you.
I like to help you?
I'm going to be.
You didn't want them.
I thought you didn't do that.
We don't want me.
Tom didn't like anything.
Tom and Mary wanted to do that.
Tom wasn't here?
I got him.
Tom is going to do that the time.
Mary didn't want to help.
I need to do that?
Why are you and you have?
I'd like that.
It's not you.
That's the last meeting.
I don't want to tell Tom and Mary like it.
How did you need to go.
I need Tom.
I need to help Mary.
Tom is not to do?
Tom doesn't want to get here.
I want to do that.
Tom is going to go.
Tom is going to help us.
I know that Tom and Mary about?
We couldn't come.
I thought Tom was?
Did you get his name.
I don't say Tom was ready.
```

As it can be noticed after reading the sentences, both algorithms are working well: no failure modes (like *mode collapse*) seem to be present. However, the majority of the sentences present at least one mistake, specially the long ones.

This can be happening because of the inner constraints in the network design introduced by the fact of using gradient descent: the whole system has to be differentiable. The biggest limitation that this produces is the technical impossibility of building a *fully visible belief network* architecture in the generator, because the sampling operation is not differentiable. That forces the algorithm to directly generate (*pseudo*) *one-hot-encoding-like* representations, which is much more inaccurate and difficult than generating a probability distribution for the next-character conditioned to the previous ones: $P(c_t|c_{t-1}, c_{t-2}, c_{t-3}, ..., c_1)$. Without this feature, the model has to decide in each sequence step which characters to output and it does not allow implementing more complex heuristics (beam search [79] for example). Strictly in that sense, it would be desirable to use some *hessian-free* optimization method [80] instead of gradient descent, but it would enormously complicate the training phase.

Finally, the results from the different aspects of the language have been analyzed below.

- *Morphology*: as it can be noticed after reading the generated samples, the morphology of the words in the sentences is quite good. The algorithms learned plenty of complex aspects of the language: apostrophes, question marks, correctly use of capital letters, punctuation symbols and general use of different vocabulary. The *1-gram precision* is a good indicator of how good the morphology of the generated samples looks like.

- *Syntax*: the majority of the generated samples show a good local syntax but most of them show issues in the general view. It seems that the networks have been able to form words and combine them so that they have a good local structure. Even though, the global structure of the majority of the generated samples seems not to be preserved. *2 and 3-gram precision* are good proxies for assessing the local syntax correctness.

- *Semantics*: in this case it happens something similar as in the syntax case. The models were able to produce sentences with local sense but with a lack of global sense. There has not been found a metric for quantifying this performance but the human judgement.

Chapter 6

Conclusions

The results in section 5.3 show that the *Generative Adversarial Networks* are able to generate text. However, the optimal models have taken a long time to train and no impressive results have been achieved. Apart from this, early trials with noisier datasets (*twitter*) showed that the model performance in text generation tasks highly depends on the quality of the data. Compared to other tasks like image and audio generation, in which these models showed very impressive results, *GANs* seem not to be the generative model which best suits for text generation. *Ian Goodfellow* [7] stated that *GAN* models usually struggle with discrete data generation and proposed using algorithms to train them which do not require gradient computations nor differentiability of the model function: for example the *reinforce* algorithm [81] or other typical algorithms broadly known in the reinforcement learning field [82]. Still, it can be concluded that the *GAN* algorithms should not be the choice when a text generation task arises.

From the architectures tested during this work, a set of take away conclusions have been compiled and described below. Not all these conclusions were expected, but they are result of the experimentation.

- Reducing the noise signal of the *minibatches* is highly recommended. It can be done by increasing their size, so that differences between the gradients from one *minibatch* to another are minimized. It showed a meaningful improvement in the final performance of the experiments conducted during this project.

- The *convolutional neural networks* filter size showed to be a crucial parameter to tune in the performance-stability trade off. Large convolutions yielded better results while making the whole system more prone to diverge and vice-versa. Optimal convolution sizes ranged from 9 to 11. A possible explaination of this may be the fact that the convolution sizes control the complexity of the model. Large complexities tend to produce *overfitting* while small ones lead to *underfitting*.

- Increasing the number of recurrent cells in the recurrent modules substantially improved the quality of the generations. This may happen because the more number of cells a network has, the more predicting power and memory is it provided with. In this case, large amounts of memory are required in order to make the model able to correctly model the complex structures of the language.

The results can be generalized to all kinds of *Generative Adversarial Networks* adjusted with gradient based optimization methods, because even having achieved very stable models (at least with both *CNN* and *RNN* architectures), the resulting models do not show very good results in comparison to other methods (*char-rnn* for instance).

In the research and development of this work, some *GAN* versions trained using reinforcement learning algorithms and showing good results have appeared in the bibliography [83, 84, 85, 86, 87, 88]. Still, other models like *variational autoencoders* perform well without such a complex setting.

Nevertheless, a potential *production-ready* application for the architectures developed in this work could be a keyword generation system for specific domains. As they represent short pieces of text which may not be gramatically correct, the approaches described in this work could perform well. A good example could be generating novel keywords for increasing the variability of them in online marketing campaigns. In this field, vast amounts of keywords are used in order to try to cover a whole field maximizing the *market share*[1]. This task is typically done manually, with the help of some automation tools. These algorithms could ingest a big amount of existing keywords and generate new ones, strongly related with the existing ones, but different.

[1]understood as the proportion of the *keywords market* which is owned by the user with respect to the competence

Chapter 7

Next steps

A set of next steps has been identified in order to provide the reader with ideas on how to continue this research line, in case he/she is interested. These ideas have been collected during the research and development of this work, and only those which have been partially tested or even not tested (and seem to be promising to the intuition of the author) have been included here.

- The use of a *recurrent neural network* in the discriminator seems to be very appropriate due to the sequential nature of text. However, it would require the model not to apply the gradient penalty in the *GAN* loss function; because it implies a second derivative to be calculated. As explained in section 4, there is a technical limitation in the main frameworks when performing this task. That is why it could be a good idea to start from the *WGAN* definition and try to improve the way it enforces the *Lipschitz* constraint avoiding using second-order derivatives calculations. A good example of this effort is the *SLOGAN* [89] definition but it did not perform well in practice. That is why a soft version of the weight clipping is proposed. Instead of clipping the weights of the network cutting the extreme values to fit in a given range, the following function could be used: $\mathbf{W} \leftarrow c \cdot \tanh(\mathbf{W})$.
- The *Gumbel softmax* seems to be a promising way of turning the sampling operation into a differentiable function, which would be useful for using a *recurrent neural network* in the discriminator while applying a gradient penalty in the loss function. It has been tested in this work and it did not produced good results, but further tests should be done in order to fully discard this alternative.
- *Reinforcement learning* optimization algorithms can be a good way of overcoming the problem, because it removes the differentiability constraint in the network and fully visible belief networks could be used.
- The algorithms proposed in this work should be tested using much shorter texts, such as keywords. The hypothesis in this scenario is that the *re-*

current neural network would produce dramatically better results because the error which is propagated in the *pseudo-sampling* operation that the network is currently forced to do would decrease with the length of the sentences generated. This error must grow exponentially with the size of the generated sentence.

- Developing compatibility with second-order derivarives in the general frameworks (*Tensorflow* and *Pytorch*) would be extremely useful for this study, allowing us building sequential discriminators.
- Using *skip-gram* precision as additional evaluation metric must be desirable. It consists of calculating non-contiguous words matching proportions between the generated samples and the real *dataset*. It would enormously help quantifying the global syntax correctness of the generated sentences.

Appendices

Appendix A

Tested architectures

In the following table, a set of architecture designs are summarized. Numerical results of accuracy are not provided because not all the architecture designs in the table has been tested until optimality (because they have shown some issues in early steps of the optimization process) and some of them even didn't converge. The goal of this information is to give a general overview of all the techniques and alternatives that have been exhausted during the final architecture design and which have formed the iterative process for coming up with the best performing architectures. The versions have been enumerated chronologically so that *V1* was the first test and *V36* the last one.

The architectures for which convergence was achieved have been highlighted in **bold**. Versions 14 and 20 (highlighted in ***italics***) correspond with the *RNN* and *Convolutional* architectures described in section 4.1 and 4.2.

Version	GAN Family	Batch Size	Learning rates[1]	Description	Results
V1	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Embedding in the generator recurrent module, applied in the decoder to each output**	**The model converges but it is a bit unstable**
V2	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Generator without embedding in the recurrent module, learning rate of the generator increased x15**	**The model converges and is stable, but the discriminator seems to be too strong**
V3	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Discriminator provided with some dense layers at the end instead of using only convolutional layers and average** pooling	**The model converges and is stable, but it does not show any improvement over previous trials**

V4	WGAN-GP	32	10^{-5}, $1.5 \cdot 10^{-4}$	Convolutional blocks in the discriminator changed by inception-1d modules	The model does not converge
V5	WGAN-GP	32	10^{-5}, $2.5 \cdot 10^{-4}$	Convolutional blocks in the discriminator changed by inception-1d modules. Lipschitz constant increased to 100	The model does not converge
V6	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Add some discrete terms to the generator input latent vector in order to try to help it generate discrete distributions**	**The model converges, is stable and slightly improves previous trials**
V7	WGAN-GP	32	10^{-5}, $1.5 \cdot 10^{-4}$	Increased the size of the generator and the discriminator x4	The model does not converge
V8	WGAN-GP	32	10^{-5}, $1.5 \cdot 10^{-4}$	Increased the size of the convolution to 15 (over V7)	The model does not converge
V9	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Increased the size of the convolution to 15 (over V2)**	**The model converges, is stable and improves previous trials**
V10	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Implementation of CT-GAN over the architecture of V2**	**The model converges and is extremely stable, but does not achieve better results**
V11	**CT-GAN**	**32**	10^{-5}, 10^{-5}'	**From V10, fixed the same learning rates for the generator and the critic (0.0001) and implemented CT-GAN [90]**	**The model converges and is extremely stable, but achieves very poor results**
V12	**WGAN-GP**	**32**	10^{-5}, $1.5 \cdot 10^{-4}$	**Add some noise to the real data one hot encodding in order to make it easy for the generator to approximate the required encoding. Architecture of V2.**	**The model converges and is relatively stable, but does not achieve good results**
V13	**WGAN-GP**	**128**	10^{-5}, $1.5 \cdot 10^{-4}$	**Increased the batch size to 128 using the V2 architecture**	**The model converges and is stable. It achieves much better results**
V14	*WGAN-GP*	*512*	$\mathbf{10^{-5}}$, $\mathbf{1.5 \cdot 10^{-4}}$	*Increased the batch size to 512 using the V2 architecture*	*The model converges and is stable. It achieves even better results than V13*

V15	WGAN-GP	32	10^{-5}, $1.5 \cdot 10^{-4}$	**From V2, add 2 additional recurrent layers to try a MultiRNN setting in the generator**	**The model converges and is stable. It does not improve previous trials**
V16	CT-GAN	256	10^{-5}, $1.5 \cdot 10^{-4}$	Implementation of CT-GAN over the architecture of V4 (with inception modules in the discriminator)	The model does not converge
V17	**WGAN-GP**	**256**	10^{-5}, $1.5 \cdot 10^{-4}$	**Removed the feed-forward connections in the recurrent module of the generator, over V2**	**The model converges and is relatively stable, but does not show any improvement over the previous trials**
V18	**WGAN-GP**	**256**	10^{-5}, $1.5 \cdot 10^{-4}$	**Starting from V17, changed the LSTM cells by GRU cells**	**The model converges and is relatively stable, but produces worse results than the V17**
V19	**WGAN-GP**	**512**	10^{-5}, $1.5 \cdot 10^{-4}$	**Starting from V14, changed the LSTM cells by GRU cells**	**The model converges and is stable, but does not improve the V14**
V20	***WGAN-GP***	***512***	$\mathbf{10^{-5}}$, $\mathbf{1.5 \cdot 10^{-4}}$	***Starting from V14, changed the architecture of the generator by a convolutional neural network***	***The model converges and is stable. In addition, it improves significantly over V14***
V21	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Starting from V14, changed the architecture of the generator by a convolutional+recurrent neural network with a context vector as initial state. Increased the size of the discriminator network.	The model does not converge
V22	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Starting from V14, changed the architecture of the generator by a convolutional+recurrent neural network with a zeros vector as initial state. Increased the size of the discriminator network.	The model does not converge
V23	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Starting from V1, add a softmax in the output of each cell and multiplied the softmax input by 10 in order to produce sharper encodings	The model does not converge

V24	**WGAN-GP**	**512**	10^{-5}, $1.5 \cdot 10^{-4}$	**Starting from V1, add a softmax in the output of each cell**	**The model converges and produces better results than V1, but they are worse than the previous trials**
V25	WGAN-GP	512	$5 \cdot 10^{-5}$, $1.5 \cdot 10^{-4}$	Starting from V21, increased the generator learning rate by 5	The model does not converge
V26	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Implemented a Gumbel softmax in the generator recurrent module, at the end of each cell	The model does not converge
V27	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Starting from V26, removed the softmax operation	The model does not converge
V28	WGAN-GP	512	$5 \cdot 10^{-5}$, $1.5 \cdot 10^{-4}$	Starting from V26, fixed equal learning rates	The model does not converge
V28b	WGAN-GP	512	$5 \cdot 10^{-5}$, $1.5 \cdot 10^{-4}$	Starting from V28, switched the generator architecture to a MultiRNN and implemented an annealing of the Gumbel's Softmax tao parameter	The model does not converge
V29	WGAN-GP	512	10^{-5}, $1.5 \cdot 10^{-4}$	Starting from V26, add dense layers at the end of the discriminator	The model does not converge
V31	**WGAN-GP**	**128**	10^{-5}, $1.5 \cdot 10^{-4}$	**Test with pytorch, V14 similar implementation**	**The model converges and produces good results, a bit worse than V14**
V32	**WGAN-GP**	**512**	10^{-5}, $1.5 \cdot 10^{-4}$	**V14 applying some changes to the synthetic instances used to apply the gradient penalty: now interpolating over the batch dimension only**	**The model converges but does not produce better results than the previous trials**
V33	**WGAN-GP**	**128**	10^{-4}, $1.5 \cdot 10^{-3}$	**V14 applying some changes to the synthetic instances used to apply the gradient penalty: now interpolating over the batch dimension 0 and 2**	**The model does not converge**
V34	**WGAN-GP**	**128**	10^{-5}, 10^{-5}	**V31 with equal learning rates**	**The model converges but produces worse results than V31**

V35	SLOGAN	64	10^{-5}, 10^{-5},	SLOGAN implementation[2] [89] in order to use an RNN-RNN architecture (RNN for the generator and RNN for the critic). Only using the last sequence step in the critic	The model does not converge
V36	SLOGAN	512	10^{-5}, 10^{-5},	SLOGAN implementation in order to use an RNN-RNN architecture (RNN for the generator and RNN for the critic). Using all the sequence steps in the critic	The model does not converge

[1]first, for the generator and second for the critic

[2]unpublished implementation of a *WGAN* alternative without using gradient penalty terms in the loss function

Bibliography

[1] T. Karras, T. Aila, S. Laine, and J. Lehtinen, “Progressive growing of gans for improved quality, stability, and variation,” in *NIPS 2017 conference*, 2017.

[2] A. J. Cawsey, B. L. Webber, and R. B. Jones, “Natural language generation in health care,” *Journal of the American Medical Informatics Association*, vol. 4, pp. 473–482, Nov. 1997.

[3] O. Rambow, S. Bangalore, and M. Walker, “Natural language generation in dialog systems,” in *Proceedings of the First International Conference on Human Language Technology Research*, HLT ’01, (Stroudsburg, PA, USA), pp. 1–4, Association for Computational Linguistics, 2001.

[4] D. Z. Inkpen and G. Hirst, “Near-synonym choice in natural language generation,” in *In Proceedings of the International Conference RANLP-2003 (Recent Advances in Natural Language Processing)*, pp. 4–3, John Benjamins Publishing Company, 2003.

[5] R. Lebret, D. Grangier, and M. Auli, “Neural text generation from structured data with application to the biography domain,” in *Empirical Methods in Natural Language Processing (EMNLP)*, Nov. 2016.

[6] D. Wulf and V. Bertsch, “A natural language generation approach to support understanding and traceability of multi-dimensional preferential sensitivity analysis in multi-criteria decision making,” *Expert Systems with Applications*, vol. 83, pp. 131 – 144, 2017.

[7] I. GoodFellow, “Generative adversarial networks for text (reddit discussion).” `https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/`.

[8] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems 27* (Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, eds.), pp. 2672–2680, Curran Associates, Inc., 2014.

[9] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *Proceedings of the 34th International Conference on Machine Learning* (D. Precup and Y. W. Teh, eds.), vol. 70 of *Proceedings of Machine Learning Research*, (International Convention Centre, Sydney, Australia), pp. 214–223, PMLR, Aug. 2017.

[10] N. J. Nilsson, *The Quest for Artificial Intelligence.* New York, NY, USA: Cambridge University Press, 1st ed., 2009.

[11] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[12] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *London Mathematical Society*, vol. 2, no. 42, pp. 230–265, 1936.

[13] A. Hodges, *Alan Turing: The Enigma.* Walker & Company, 2000.

[14] A. M. Turing, "Computing Machinery and Intelligence," *Mind*, vol. LIX, pp. 433–460, 1950.

[15] J. Lighthill, "Artificial intelligence: A general survey," *Artificial Intelligence: a paper symposium, Science Research Council*, 1973.

[16] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach.* Pearson Education, 2 ed., 2003.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.

[18] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, pp. 58–68, Mar. 1995.

[19] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, April 1965.

[20] M. Lipshutz, R. McEntire, and D. P. McKay, "Lima: a logistics inventory management assistant," *The Seventh IEEE Conference on Artificial Intelligence Application*, vol. I, pp. 393–397, Feb 1991.

[21] M. Benaroch and V. Dhar, "An intelligent assistant for financial hedging," in *The Seventh IEEE Conference on Artificial Intelligence Application*, vol. I, pp. 168–174, Feb 1991.

[22] L. Johnson and A. Hoback, "From prototype to production: expanding expert systems project management planning," in *IEEE/ACM International Conference on Developing and Managing Expert System Programs*, pp. 286–294, Sep 1991.

[23] T. Falas, A. Charitou, and C. Charalambous, “The application of artificial neural networks in the prediction of earnings,” in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence*, vol. 6, pp. 3629–3633, Jun. 1994.

[24] M. G. Dorrer, A. N. Gorban, and V. I. Zenkin, “Neural networks in psychology: classical explicit diagnoses,” in *The Second International Symposium on Neuroinformatics and Neurocomputers*, pp. 281–284, Sep 1995.

[25] E. Denby and J. Gammack, “The naming of colours: investigating a psychological curiosity using AI,” in *Neural Information Processing Systems, 1999*, vol. 3, pp. 964–973, 1999.

[26] T. Ogawa, T. Minohara, H. Kanada, and Y. Kosugi, “A neural network model for realizing geometric illusions based on acute-angled expansion,” in *Neural Information Processing, 1999. Proceedings. ICONIP ’99. 6th International Conference on*, vol. 2, pp. 550–555, 1999.

[27] L. I. Perlovsky, “Emotions, learning and control,” in *Proceedings of the 1999 IEEE International Symposium on Intelligent Control Intelligent Systems and Semiotics*, pp. 132–137, 1999.

[28] N. DeClaris, “A systems approach to medical decision aiding,” in *Conference Proceedings 1991 IEEE International Conference on Systems, Man, and Cybernetics*, vol. 3, pp. 2103–2107, Oct. 1991.

[29] D. A. Klein and E. H. Shortliffe, “Interactive diagnosis and repair of decision-theoretic models,” *The Seventh IEEE Conference on Artificial Intelligence Application*, vol. I, pp. 289–293, Feb 1991.

[30] W. F. Punch, “Large interactions of compiled and causal reasoning in diagnosis,” *IEEE Expert*, vol. 7, pp. 28–35, Feb 1992.

[31] A. Cinar, E. Tatura, J. DeCicco, R. Raj, N. Aggarwal, M. Chesebro, J. Evans, M. Shah-Khan, and A. Zloza, “Automated patient monitoring and diagnosis assistance by integrating statistical and artificial intelligence tools,” vol. 2, p. 700, Oct 1999.

[32] T. Smithers, M. X. Tang, and N. Tomes, “An approach to intelligent drug design support,” in *[1993] Proceedings of the Twenty-sixth Hawaii International Conference on System Sciences*, vol. I, pp. 634–645, Jan 1993.

[33] J.-H. Yoo, B.-H. Kang, and J.-U. Choi, “A hybrid approach to auto-insurance claim processing system,” vol. 1, pp. 537–542, Oct 1994.

[34] H. M. Mashaly, A. M. Sharaf, M. M. Mansour, and A. A. El-Sattar, “Implementation of an artificial neural network based controller for a photovoltaic energy scheme,” vol. 4, pp. 2545–2549, Jun 1994.

[35] T. Koyama, T. Horie, T. Yoshioka, F. Yoshitani, and J. Takahashi, "A highly intelligible speech synthesis for banking services in financial network system anser," in *IVTTA: Interactive Voice Technology for Telecommunications Applications. IEEE 4th Workshop*, pp. 87–90, Sep 1998.

[36] M. Campbell, A. J. H. Jr., and F. hsiung Hsu, "Deep blue," *Artificial Intelligence*, vol. 134, pp. 57–83, 2002.

[37] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, "Mastering the game of go without human knowledge," in *Nature*, vol. 550, pp. 354–359, Oct 2017.

[38] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," Dec 2017.

[39] A. Rahimi, "Nips 2017, test-of-time award presentation," in *NIPS 2017*, Presented in the NIPS conference as a Test-of-time award presentation, Long Beach, CA, 2017.

[40] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[41] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[42] B. Widrow, "An adaptive 'Adaline' neuron using chemical 'memistors'," *technical report at Stanford university, Solid-State Electronics Laboratory, under an Office of Naval Research contract*, 1960.

[43] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry.* Cambridge, MA, USA: MIT Press, 1969.

[44] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," vol. 36, pp. 193–202, Feb 1980.

[45] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *National Academy of Sciences of the United States of America*, vol. 79, pp. 2554–2558, Apr. 1982.

[46] P. Smolensky, "Parallel distributed processing: Explorations in the microstructure of cognition," ch. Information Processing in Dynamical Systems: Foundations of Harmony Theory, pp. 194–281, Cambridge, MA, USA: MIT Press, 1986.

[47] S. Kullback and R. A. Leibler, "On information and sufficiency," *Ann. Math. Statist.*, vol. 22, no. 1, pp. 79–86, 1951.

[48] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, pp. 1735–1780, Nov. 1997.

[49] "A survey on the application of recurrent neural networks to statistical language modeling," *Computer Speech and Language*, vol. 30, no. 1, pp. 61 – 98, 2015.

[50] F. M. Bianchi, E. Maiorino, M. C. Kampffmeyer, A. Rizzi, and R. Jenssen, "An overview and comparative analysis of recurrent neural networks for short term load forecasting," *CoRR*, vol. abs/1705.04378, 2017.

[51] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, Nov 1998.

[52] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," *NIPS*, 2007.

[53] G. Tesauro, "Practical issues in temporal difference learning," *Machine Learning*, vol. 8, pp. 257–277, May 1992.

[54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *25th International Conference on Neural Information Processing Systems*, vol. 1 of *NIPS'12*, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[55] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, 2012.

[56] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.

[57] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proceedings of Workshop at ICLR 2013*.

[58] D. Wu and X. Liu, "Improve training stability of semi-supervised generative adversarial networks with collaborative training," in *Proceedings of ICLR 2018*, Sep. 2017.

[59] I. J. Goodfellow, "NIPS 2016 tutorial: Generative adversarial networks," *Neural Information Processing Systems*, Dec. 2016.

[60] B. Fuglede and F. Topsoe, "Jensen-shannon divergence and hilbert space embedding," in *International Symposium on Information Theory, 2004*, pp. 31–, June 2004.

[61] B. J. Frey, G. E. Hinton, and P. Dayan, "Does the wake-sleep algorithm produce good density estimators?," in *Advances in Neural information Processing Systems*, pp. 661–667, MIT Press, 1996.

[62] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," *Proceedings of the 28th International Conference on Machine Learning (ICML-11), Bellevue, Washington, USA*, pp. 1017–1024, Jan. 2011.

[63] A. Graves, "Generating sequences with recurrent neural networks," *Department of Computer Science, University of Toronto, CoRR*, 2013.

[64] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel recurrent neural networks," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA*, Jun. 2016.

[65] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," The International Conference on Learning Representations (ICLR), Banff, Dec. 2013.

[66] C. Doersch, "Tutorial on variational autoencoders," *Technical Report. Carnegie Mellon / UC Berkeley*, 2016.

[67] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein, "Unrolled generative adversarial networks," in *proceedings of ICLR*, 2017.

[68] B. Poole, A. A. Alemi, J. Sohl-Dickstein, and A. Angelova, "Improved generator objectives for gans," *NIPS Workshop on Adversarial Learning, Google AI*, 2016.

[69] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training gans," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, (USA), pp. 2234–2242, Curran Associates Inc., 2016.

[70] M. Arjovsky and L. Bottou, "Towards Principled Methods for Training Generative Adversarial Networks," in *proceedings of ICLR*, Jan. 2017.

[71] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved Training of Wasserstein GANs," *Advances in Neural Information Processing Systems (NIPS)*, 2017.

[72] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, (Cambridge, MA, USA), pp. 3104–3112, MIT Press, 2014.

[73] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 448–456, JMLR.org, 2015.

[74] Z. Zhou, W. Zhang, and J. Wang, "Inception score, label smoothing, gradient vanishing and -log(d(x)) alternative," *CoRR*, Aug. 2017.

[75] A. Borji, "Pros and cons of GAN evaluation measures," *Center for Research in Computer Vision, University of Central Florida, Orlando, FL, USA, CoRR*, Feb. 2018.

[76] S. Barratt and R. Sharma, "A note on the inception score," *Department of Electrical Engineering, Stanford University, Stanford, CA, USA*, Jan. 2018.

[77] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation," in *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, (Stroudsburg, PA, USA), pp. 311–318, Association for Computational Linguistics, 2002.

[78] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[79] M. Freitag and Y. Al-Onaizan, "Beam search strategies for neural machine translation," in *Proceedings of the First Workshop on Neural Machine Translation, pages 56–60, Vancouver, Canada*, Aug. 2017.

[80] J. Martens and I. Sutskever, *Training Deep and Recurrent Networks with Hessian-Free Optimization*, pp. 479–535. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.

[81] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning.* Cambridge, MA, USA: MIT Press, 1st ed., 1998.

[82] C. Szepesvári, *Algorithms for Reinforcement Learning.* Morgan & Claypool, 2010.

[83] Y. Zhang, Z. Gan, K. Fan, Z. Chen, R. Henao, D. Shen, and L. Carin, "Adversarial Feature Matching for Text Generation," Jun. 2017.

[84] Y. Zhang, Z. Gan, and L. Carin, "Generating Text via Adversarial Training," in *Workshop on Adversarial Training, NIPS 2016, Barcelona, Spain*, 2016.

[85] J. Guo, S. Lu, H. Cai, W. Zhang, Y. Yu, and J. Wang, "Long Text Generation via Adversarial Training with Leaked Information," *CoRR*, Sep. 2017.

[86] W. Fedus, I. Goodfellow, and A. M. Dai, "MaskGAN: Better Text Generation via Filling in the ______," in *Proceedings of ICLR 2018*, Jan. 2018.

[87] Z. Hu, Z. Yang, X. Liang, R. Salakhutdinov, and E. P. Xing, "Toward Controlled Generation of Text," in *Proceedings of Machine Learning Research*, Mar. 2017.

[88] S. Rajeswar, S. Subramanian, F. Dutil, C. Pal, and A. Courville, "Adversarial Generation of Natural Language," *CoRR*, May. 2017.

[89] Lernappart, "More improved training of wasserstein gans and dragan." `https://lernapparat.de/more-improved-wgan/`.

[90] X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang, "Improving the improved training of wasserstein gans: A consistency term and its dual effect," *CoRR*, Mar. 2018.