POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

# Automated Content Generation with Semantic Analysis and Deep Learning

**Supervisor:**
Prof. Maurizio Morisio

**Candidate:**
Thai Hao Marco VAN

**Internship Tutor**
Dr. Giuseppe Rizzo

Academic Year 2017-2018

# Contents

# List of figures

# List of tables

# Chapter 1

# Introduction

Natural language is the basis of how humans communicate and understand each other. The fundamental skills required in human communication are the ability to understand the language and the ability to use the language. These are natural tasks for humans but are considered as one of the most challenging problems to be solved by machinery.

Natural Language Processing (NLP), also known as computational linguistic, is the research field that combines linguistic and artificial intelligence to study and analyze the human language. The goal is to develop systems to enable machines to communicate with humans making these believe they are interacting with other humans. During its evolution natural language has been approached in several ways.

Rule-based algorithms were commonly used at the beginning. They require to define a set of rules to handle the characteristics of human language. Despite being easy to interpret the amount of rules that has to be hand-crafted is very large, so it is not efficient. Also it is evident that this approach has a lot of limitations due to the fact that natural language contains a lot of ambiguities and exceptions that can lead to contradictions. An early attempt to build a system able to interact with humans was made by [61] with ELIZA. It is based on pattern matching technique, by which given an utterance from a human, the system would search for compatible pre-defined answer templates and then it replaces the placeholders. This system has no knowledge about the language so it execute commands without really understanding what have been asked. Hence, the limit of ELIZA is that it cannot handle all the questions made by the user, as a result a lot of answers are generic and repetitive.

Recent advances in the field of artificial intelligence allowed to tackle the problem with a new point of view. The new solutions are based on neural network techniques, which offer the advantage of automatically create a set of rules without the necessity to be defined explicitly. They are therefore more efficient since they learn on their own, and they can capture patterns in the data that could be hard to describe.

In the last years, the interaction with technology has increased substantially.

For this reason the necessity of having intelligent systems capable to interact with people is growing. Many tasks are getting automated and some of them involve text generation. For instance, numerous companies are using generative tools to periodically put together structured data into reports written in natural language that have to be readable by other people.

In this work, we explore the process of text generation by modelling the language with neural networks, since these have shown promising results in many applications.

## 1.1 Natural Text Generation

Natural text generation is the computational task of producing text in natural language. It has gained a lot of attention in the last years thanks to the many applications in which it can be employed. Some of the most common are neural machine translation, text summarization and dialogue generation.

Text generation is approached by building language models that aim to represent the probability distribution of the words in a text, in order to predict the next word given the words that precede it. The current state of the art language models are developed using neural networks. Even though a lot of progress has been made in this field, the most popular solutions still have limitations. There are several aspects to take into account, from syntax and semantic to how to express global meaning within the sentence. Syntax structures can be learned by statistical models, for instance how to structure a sentence with subject, predicate and object. But the main issues are related to how composing the sentence in a meaningful way, and how to maintain coherence throughout the sentence. Current approaches generate text that tends to not be coherent, in other words they start with a topic and then switch completely to another topic in the middle of the sentence.

Text generation is considered as a sequence prediction problem. The common approach for this kind of problem is to use recurrent neural networks and maximize the likelihood to predict the observed data [27]. This approach has some limits as described by [4], one is called exposure bias. During the training process the network is feeded with the training data, but during the inference time it uses its own predictions as input. As a result when the network makes a prediction mistake the error is accumulated and propagated throughout the sequence, causing the model to being unable to work properly. To address this issue [4] proposed the Scheduled Sampling technique, which consists of randomly alternating the input during the training process by feeding the real data or its own predictions, but [24] showed that this method is inconsistent.

The introduction of generative adversarial networks (GANs) [20] opened the doors to a new way of how generative tasks can approached. In the specific, the previous approaches are based on having an explicit likelihood function, whereas

GANs likelihood function is implied by having two components with two different objective function [13]. For this reason these networks have all the characteristics to alleviate the exposure bias problem. GANs were designed to deal with continuous data such as images, while natural language is based on discrete tokens and therefore a different implementation is required, this will be discussed more in the detail in the next chapter.

## 1.2 Goals

GANs have demonstrated to be successful in the image domain, also some recent implementations showed that they can be applied in the natural language domain. The goal of this work is to implement GANs to generate readable text. Before this we first explore the capabilities of the most used generative models based on neural networks. Then we focus on the generative adversarial networks (GANs) applied in the natural language domain.

The thesis works on the following research questions:

> **Research Question 1**: how effective are the generative models GAN-based?
>
> **Research Question 2**: how effective are the quantitative metrics used to evaluate the generated text?

## 1.3 Thesis Structure

The thesis is organized as follows.

In Chapter 2, we review the current state of the art algorithms used for text generation. It starts with an explanation of the text generation problem and how to model it. It follows the review of the most common generative models such as Recurrent Neural Networks and Generative Adversarial Networks.

In Chapter 3, we present the models we have used for our experiments. In detail, we try to use different discriminator implementations to test whether the results would improve.

In Chapter 4, we describe the experimental setup, this includes the datasets used for the experiment along with the metrics used for the evaluation.

In Chapter 5, we present and explain the results of the experiments. In addition we discuss about the advantages and drawbacks of the different implementations.

As conclusion, in Chapter 6, we present a summary of this work and discuss about the direction for future work.

# Chapter 2

# State of the art

The following chapter gives a background of the current approaches used for text generation. Section 2.1 and Section 2.2 describe the text generation task and language model respectively. Then, from Section 2.3 the discussion focuses on neural network based models.

## 2.1 Text Generation

As introduced in the previous chapter, text generation consists of generating human language. The ability of generating syntactically correct and meaningful text is important for many tasks. We also attribute semantic meaning to words, which sometimes can also be ambiguous. We can think of machine translation systems in which the word-by-word translation is not enough, they must take into account how the words are used and how to structure sentence in order to be meaningful in the translated language. Or text summarization where other than extracting portions from the source it must compose the summary in a concise and meaningful way. There a lot of challenges to face when we want to generate text through algorithms. Machines are good at dealing with structured data and well defined rules, on the contrary natural language has many characteristics hard to describe. So it is not trivial to build a model that can capture all the language traits.

## 2.2 Language Model

In order to approach the text generation problem, the first requirement is to approximate the human language with a language model.

A language model (LM) is defined as the probability distribution of a sequence of words, see Equation 2.1. In other words, given a text $T$ how likely is a given sentence $S$ be part of $T$.

$$w = (w_1,\, w_2,\, \dots,\, w_n) \tag{2.1}$$

$$P(w) = \prod_{t=1}^{l} P(w_t|\, w_1,\, \dots,\, w_{t-1}) \tag{2.2}$$

Equation 2.2 defines the problem of predicting the next word given the previous ones. In the specific, given a sentence $w = \{w_1,\, w_2,\, \dots,\, w_t,\, \dots,\, w_l\}$, where $w_t$ is the word at position $t$ within the sentence of length $l$, the language model is expressed as the product of the conditional probabilities of each word $w_t$ given $W_{1:t-1}$.

There are text generation problems that other than depending on the previous words they also depend on the context. Context is referred as a variable that conditions the outcome of the prediction. This kind of problems are defined with conditional language models, see Equation 2.3.

$$P(w|\, x) = \prod_{t=1}^{l} P(w_t|\, x,\, w_1,\, \dots,\, w_{t-1}) \tag{2.3}$$

$$P(w) = \prod_{t=1}^{l} P(w_t|\, w_{t-(n-1)},\, \dots,\, w_{t-1}) \tag{2.4}$$

Language modelling is considered as the core of many NLP tasks, those which require to approximate the probability distribution of an observed dataset. We can distinguish two main approaches for language modelling: statistical-based [10, 40] and neural networks based.

The common approach of statistical language models is to learn the probability distribution of n-grams ($n$ words at a time) within a corpus, see Equation 2.4. N-gram-based techniques have been quite successful in the past [31], but are limited because they are not able to generalize given a limited corpus. The main problem, as stated by [5], is the curse of dimensionality, which cause the model to being unable to manage unseen sequences. In other words, if the model is tested on different data than the training data, it will perform very poorly. On the other hand language models based on neural networks [5] are more flexible, since they can represent the words in the vector space and learn dependencies between them. Therefore they can handle cases not observed in the corpus, for this reason they perform better than statistical language models.

## 2.3 Artificial Neural Networks

Artificial neural networks [41] are system models created after the observation of how the human neural networks are structured. These models are known for their ability to learn from experience like human brain, they modify themselves through a training process and they get better as they are fed with more information about the world. The strength of these systems is the ability to describe complex phenomena by recognizing patterns in the data and deriving a related function. The basic structure of an artificial neural network consists of a set of weights and an activation function. It takes an input and use the weights to generate a signal. The neural network produces an output only if the signal reaches a threshold defined by the activation function. This behaviour allows the neural network to model nonlinear relationships between the input and the output.

The introduction of more powerful hardware, such as GPUs, has made possible to implement multiple layers of artificial neural networks. With the result of having deep neural networks able to abstract at an higher level the pattern within the data and consequently learn more complex functions, hence the name Deep Learning. As of today, these networks are used for a variety of tasks in computer vision, natural language processing, data processing, robotics etc.

## 2.4 Feedforward Networks

Feedforward networks (FFNs) are the basic category of neural networks. The main characteristic of these networks is that the information flows through the layers only in one direction, so the output depends exclusively by the input. These models have the capability to approximate any nonlinear function. To achieve this, every neuron is provided of an activation function that allows the passage of the information only if it goes above a threshold.

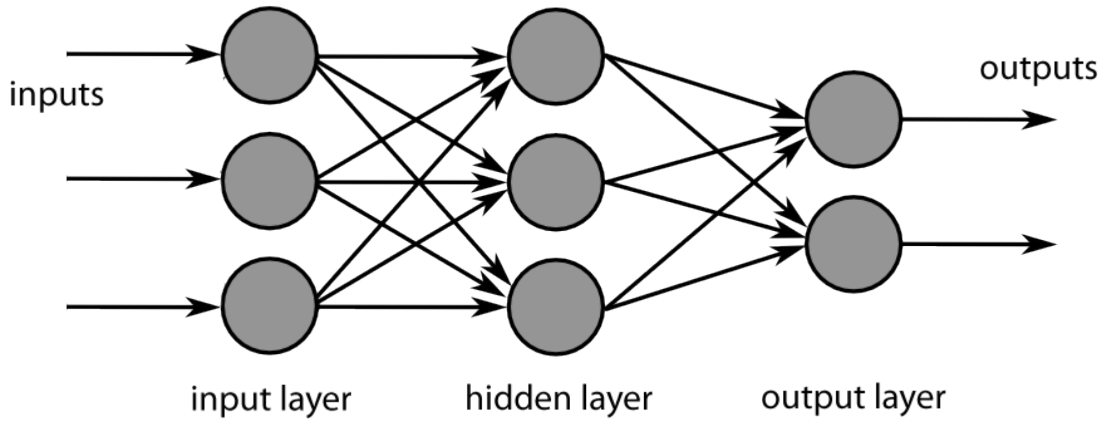


Figure 2.1. Feedforward neural network

Figure 2.1 shows the layers in a FFN, the operation process can be summarized by the formula $\hat{y} = f(W \cdot x + b)$. $x$ is the input vector with size $n \times m$ ($n$ inputs and $m$ is the single input vector representation size). $W$ is the weight matrix of the hidden layer that is learned through training. $b$ is the bias, which is a learnable value used to adjust the predicted output so that it is close to the expected output. $f$ is the activation function. $\hat{y}$ represents the predicted outputs.

### Backpropagation

FFNs learn to improve their prediction through a learning process in a supervised environment, this means the network needs to know what the expected output should be. The most common mechanism used as learning technique is called back-propagation (BP) [15]. The technique derives the changes that have to be made to the weights with the goal of minimizing the error of the prediction. FFNs training process is divided in two steps:

- **Forward propagation**: given the input $x$ the model produces an output $\hat{y}$, next the model computes the error between the predicted output $\hat{y}$ and the expected output $y$, the error also called loss is used during the backward propagation.
- **Backward propagation**: the network computes the partial derivatives of the loss with respect to the weights, also known as gradients, these derivatives are then used by the gradient descent algorithm whose objective is to minimize the loss by updating the weights.

As shown in Figure 2.2, the input flows through the network with the forward propagation. The output produced by the network is compared with the expected output. A loss function computes the error between the actual output and the expected output. This error E is backpropagated through the network using gradient descent algorithm, which consists of computing the gradients with respect to the network weights.

### Overfitting problem

Training a neural network on a limited dataset could lead to the problem of overfitting, which means that the network has adapted too much to the training data and therefore is unable to handle correctly unseen data. In short, the model has memorized the training data instead of learning the underlying pattern. To overcome this problem, [54] has introduced the Dropout technique. The idea is to randomly disable some neural units, in order for them to be not too much dependent on each other. Consequently the network improve its ability to generalize.

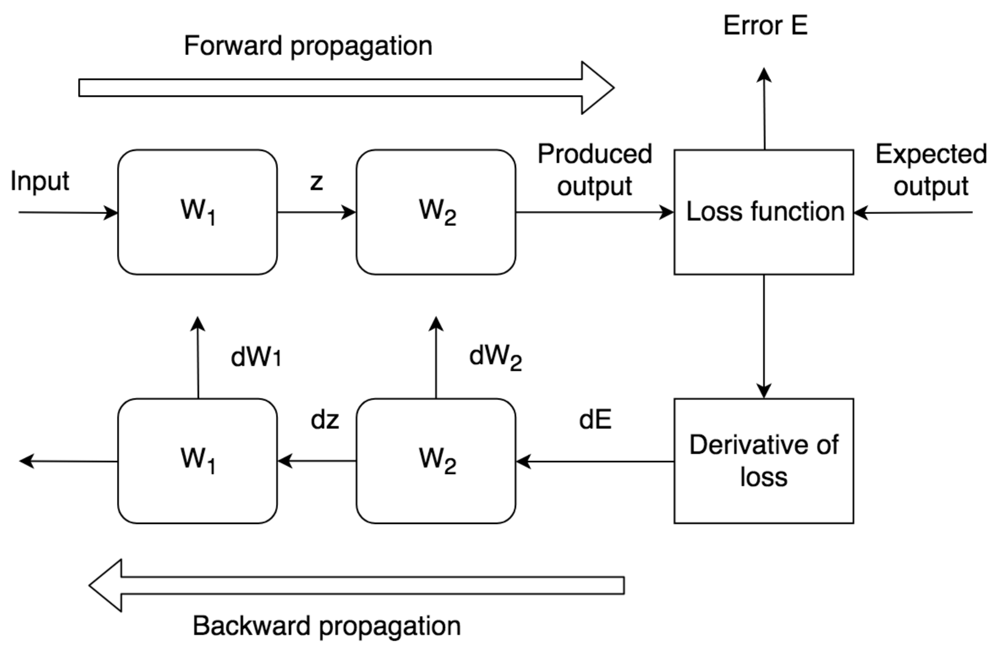


Figure 2.2. Feedforward neural network training process

### Application in Text Generation

In early stages neural language models were learned through feedforward networks [46, 1]. FFNs applied to the task of text generation learn the probability distribution over a sequence of words by taking into account a fixed length context, in short the prediction depends on the previous $n-1$ words, where $n$ is the context size. The noticeable drawback is that it lacks of a mechanism that can catch dependencies between the present and the past and keep a memory of it. This issue is addressed by the recurrent neural networks, which will be explained in the next section.

### Softmax

The $softmax$ function is important when working with discrete data such as words. It is used to represent a probability distribution over a set of classes, so that the sum of the probabilities is equal to 1. It is often used as the last layer of the network to determine the probability of the outputs. Then in order to select the most likely output over the distribution the $argmax$ operation is used.

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=0}^{k} e^{x_j}} \; for \; i = 0,\, 1,\, \ldots,\, k \tag{2.5}$$

## 2.5 Recurrent Neural Networks

Recurrent neural networks (RNN) [16] are an extension over feedforward networks. Their main advantages over FFNs are the ability to handle data of variable length and to deal with time sequence problems. To achieve this, they introduce a loop in the network that allows the neurons to be fed with the output of the previous timestep, making the current output not only be dependent by the input but also by the previous ones. In order to remember the previous input values they are provided of a hidden state, which is used as a memory that keeps all the information about the past.

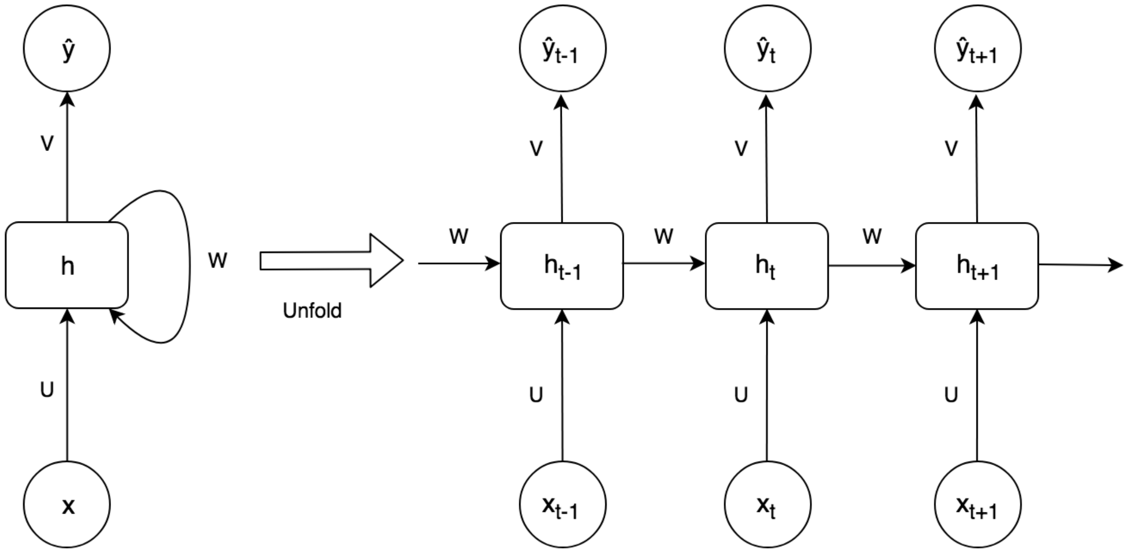


Figure 2.3. Recurrent neural network and the unfolding operation.

Figure 2.3 shows the RNN structure, in the specific the unfolding operation, which is how the network process an input stream of data. As shown, given as input a sequence $x = \{x_1, x_2, \dots, x_T\}$, where $x_t$ corresponds to a word in input at timestep $t$, the network state $h_t$ is updated using the input $x_t$ and the previous hidden state $h_{t-1}$ using the following formula $h_t = (W \cdot h_{t-1} + U \cdot x_{t-1})$, then the output distribution is generated as $\hat{y}_t = softmax(V \cdot h_t)$.

### Backpropagation Through Time

In RNNs the training process is called Backpropagation Through Time (BPTT) [62]. The backpropagation is done in two steps: the network during the unrolling process make a prediction at each timestep, so the errors between the predicted and the expected output are computed. The errors are then accumulated during the timesteps and used to update the network weights.

**Application in Text generation**

RNNs achieve particularly good results in sequence prediction, which make them ideal to be used for natural text generation [27, 43, 56, 44]. RNNs predict the next word by taking into account the history of previous seen words. The ability to remember the input history is very important: for instance consider the phrase: "He is from Italy. His first language is _", in order to correctly fill the blank _ the model has to recall the previous words to get a hint on which language to use, in this case "Italian".

Text generation task is considered as a multiclass prediction problem, where at each timestep the model has to estimate the probability distribution of the words in the vocabulary that fit the observed data. The vocabulary is defined as the set of unique word within the data. Given a corpus of $T$ words $w = \{w_1, \ \ldots, w_T\}$, the training process consists of minimizing, at each timestep $t$, the cross-entropy loss between the predicted output distribution $\hat{y}^{(t)}$ and the expected output $y^{(t)}$, as defined in Equation 2.6. $V$ represents the vocabulary.

$$H(y^t, \hat{y}^t) = -\sum_{i}^{|V|} y_i^t \log \hat{y}_i^t \tag{2.6}$$

Recurrent Neural Networks have several implementations, the most commons are RNN, LSTM and GRU. These implementations are explained in the next sections.

### 2.5.1 Vanilla RNN

Vanilla RNN is the basic implementation. It has two learnable matrix weights $U$ and $W$, and a learnable bias $b$. It takes the single input $x_t$ and the previous hidden state $h_{t-1}$ as input, then it produces the new state $h_t = tanh(U \cdot x_t + W \cdot h_{t-1} + b)$. The state $h_t$ can be used to produce a probability distribution $\hat{y}_t$ by applying the $softmax$ operation.

This implementation is not used in practice. The reason is due to the backpropagation process, in which the gradients are multiplied through the timesteps making them susceptible to the vanishing gradient problem [22, 6] or the exploding gradient problem.

**Vanishing gradient**: the values of the gradients become so small that get near to 0, therefore the weights will not be updated and the model stops to learn. This issue can be solved with the LSTM cell implementation discussed in the next section.

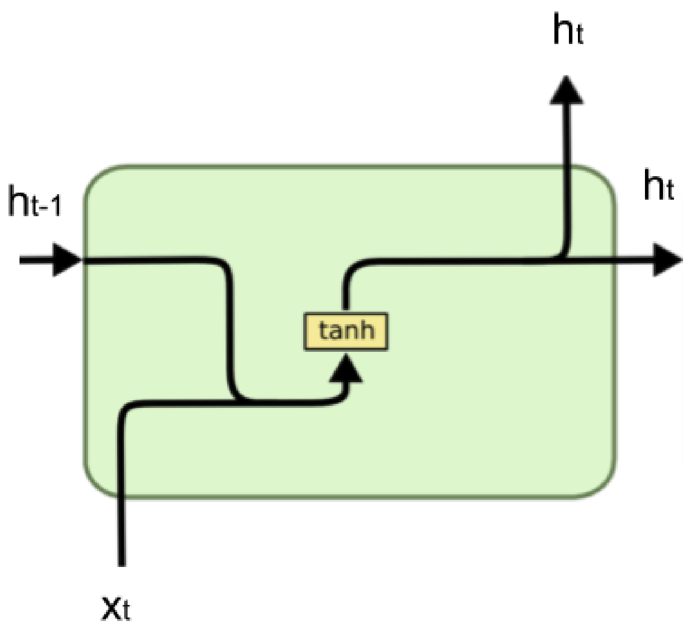


Figure 2.4. Vanilla RNN

**Exploding gradient**: in this case the gradients become very large causing the network to make unexpected updates, with the consequence that the model is not going to converge. This issue is alleviated by using the LSTM cell or can be addressed with the gradient clipping technique, which consists of limiting the gradient to a certain threshold.

### 2.5.2 Long-Short Term Memory

Long-Short Term Memory (LSTM) [23] implementation is an improvement over vanilla RNN. It prevents the vanishing gradient problem by applying a gating mechanism [18], allowing the model to learn long-term dependencies. LSTM gating mechanism consists of three gates which select the data that should be forgotten and the data that should be learned or updated in the memory.

The LSTM memory differs from the RNN implementation. It consists of the cell state ($C$) and hidden state ($h$). The former is the memory that contains the information about the past, whereas the latter contains the information used to produce the output.

[1]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

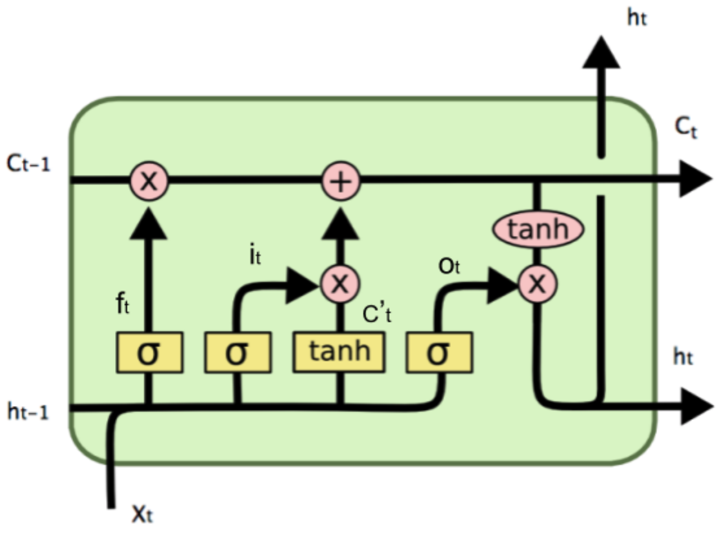


Figure 2.5. LSTM gating mechanism. Source[1]

$$C_t^{'} = tanh(W_c \cdot [h_{t-1},\, x_t] + b_c) \tag{2.7}$$

$$i_t = \sigma(W_i \cdot [h_{t-1},\, x_t] + b_i) \tag{2.8}$$

$$f_t = \sigma(W_f \cdot [h_{t-1},\, x_t] + b_f) \tag{2.9}$$

$$C_t = f_t * C_{t-1} + i_t * C_t^{'} \tag{2.10}$$

$$o_t = \sigma(W_o \cdot [h_{t-1},\, x_t] + b_o) \tag{2.11}$$

$$h_t = o_t * tanh(C_t) \tag{2.12}$$

The LSTM operations can be divided into 4 stages:

- First stage, candidate vector creation, a *tanh* layer squash the input between $-1$ and 1 creating a vector of new candidate values $C_t^{'}$ [2.7] that could be added into the memory.

- Second stage, select relevant information, the input gate $i_t$ [2.8] decides which information from the input are going to be added into the memory. The candidate values are multiplied by the input gate, which consists of a *sigmoid* layer that output values between 0 and 1.

- Third stage, forget irrelevant information, the forget gate $f_t$ [2.9] decides which information from the previous hidden state are going to be discarded from the previous state, this also uses a *sigmoid* activation. $t$ produces a new state $C_t$.

- Final step, output relevant information, the output gate $o_t$ [2.11] decides what values from our cell state $C_t$ are going to be part of the hidden state $h_t$ [2.12].

The reason why vanishing gradient does not occur in LSTM can be found in the Equation [2.10]. The cell state in this case has as activation function the identity function, therefore the derivative is equal to one. Since the term multiplied by $C_{t-1}$ is $f_t$, when the forget gate is equal to 1, the information from $C_{t-1}$ can pass through unchanged.

### 2.5.3 Gated Recurrent Unit

Gated recurrent unit (GRU) [12] is also an improvement over vanilla RNN. GRUs are similar to LSTMs because they are also provided of a gating mechanism, which allows to address the vanishing gradient problem as well. A noticeable difference from LSTM is that the memory is only managed by the hidden state, so there is no cell state. This implementation therefore allows to reduce the number of parameters.

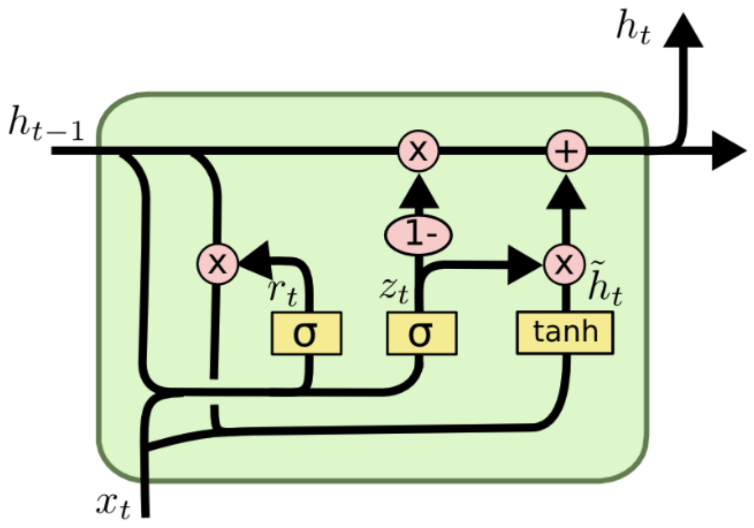


Figure 2.6. GRU gating mechanism. Source [2]

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{2.13}$$

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \tag{2.14}$$

$$\widetilde{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t]) \tag{2.15}$$

[2]http://colah.github.io/posts/2015-08-Understanding-LSTMs/

$$r_t = \sigma(W_r[h_{t-1}, x_t]) \tag{2.16}$$

The gating mechanism, unlike LSTM, is composed of two gates:

- Update gate: given the new input, it selects which information from the previous hidden state should be passed through. The result is then squashed between 0 and 1 with *sigmoid* activation function [2.13].

- Reset gate: given the new input, it decides which information from the previous hidden state should be forgotten [2.14].

The hidden state is updated by selecting which information should be forgotten [2.7] and then by selecting the information that should be used for future reference [2.16].

### 2.5.4 Independently RNN

Independently RNN (IndRNN) is an implementation proposed by [36]. As the author points out all the neurons in a RNN are fully connected, so their behaviour is hard to interpret, in particular it is not trivial to understand the roles of each neuron. To address this problem the neurons in an IndRNN cell are independent from each other and they are connected only to themselves, making each neuron have its own history.

This structure provides multiple advantages such as:

- Each neuron is specialized on a specific task since they are independent to each other.

- Ability to stack a large number of layers because they do not receive "noise" from other neurons.

- They can work with non-saturated activation function such as *ReLU*, therefore preventing the vanishing gradient problem.

$$h_t = \sigma(W \cdot x_t + U * h_{t-1} + b) \qquad h_t = \sigma(W \cdot x_t + U \cdot h_{t-1} + b) \tag{2.17}$$

The Equation [2.17] shows how the hidden state in IndRNN is computed. The weight matrix $U$ is multiplied with the previous hidden state $h_{t-1}$ using the element-wise operation $(*)$ instead of the dot product.

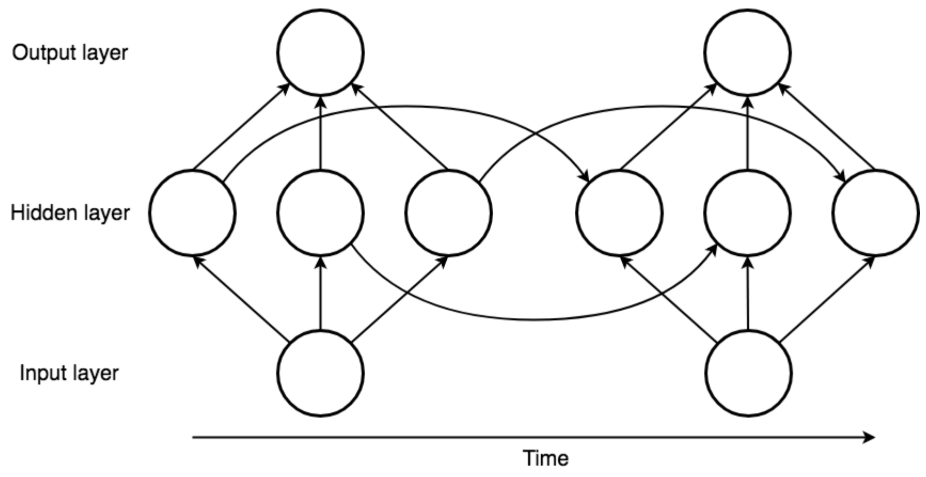


Figure 2.7. IndRNN structure
As shown the neurons within the hidden layer are connected only to themselves over the timesteps

### 2.5.5 Attention Mechanism in RNN

The attention mechanism, introduced by [3], is a technique based on the visual attention of humans, which can be explained as being able to focus on a particular part of a view while ignoring the surrounding. Hence, the basic idea of the mechanism is to pay "attention" only on a relevant part of the input in order to select the appropriate output.

The first implementation has been made by [3] for the neural machine translation, which can be defined as a conditional language modelling task [45]. A machine translation model is based on the Sequence to Sequence (Seq2Seq) architecture [57]. It has two components an encoder and a decoder. The encoder, based on RNN, is responsible of generating a hidden state that have all the information of the phrase in the source language encoded. The decoder, also based on RNN, will perform the translation by decoding the encoder hidden state. The drawback of this system is that the encoder hidden state could be cause of bottleneck, since all the information have to be compressed in a sense. The attention mechanism gives support to the decoder by providing an alignment system that keeps track of the position of the word being currently translated.

Figure [2.8] shows an example of the attention mechanism. It helps the RNN to make the next prediction by accessing all the previous states of the network. In this way the network other than having access to the current state it can also directly use the past to get more relevant information, since using only the current state can be cause of bottleneck. In the example, for the task of sentiment analysis, the network puts more emphasis on relevant part of the sentence in order to predict its

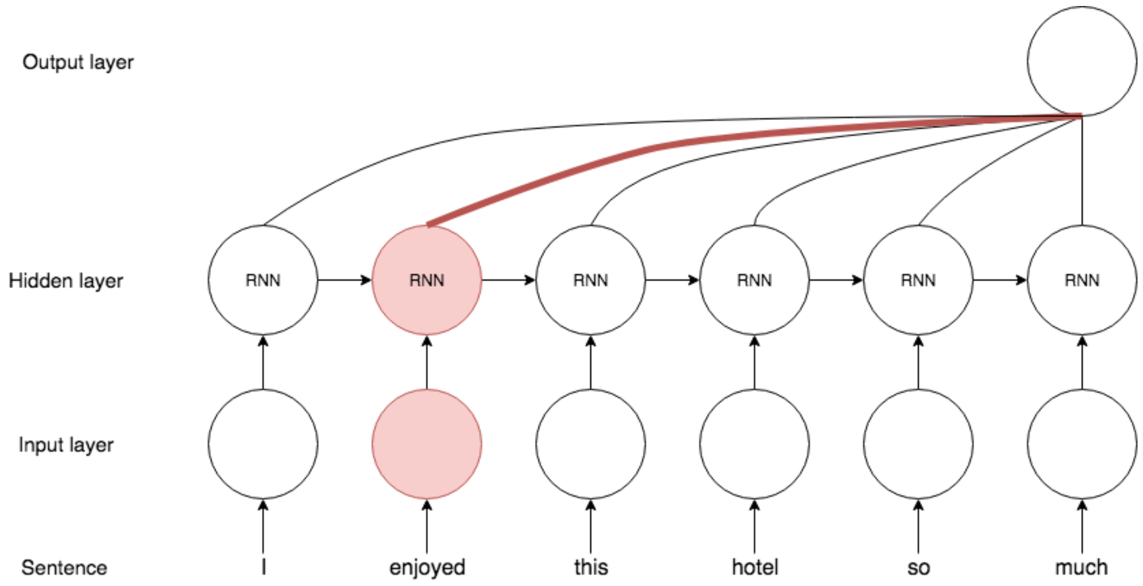


Figure 2.8. Attention mechanism in RNN: example of sentiment analysis.

sentiment. In that case by focusing on “enjoyed”, the network is able to classify with high confidence the sentence as positive.

### 2.5.6 Bidirectional

RNN models are often implemented in a forward setting, so they read the input sequence starting from the first symbol to the last one. However, for some tasks it could be important for the model to know the future state as well. For this reason Bidirectional RNN (Bi-RNN) was introduced by [52]. The network is provided of two hidden states of opposite direction, one forward and one backward. This structure allows to capture more information and is commonly in machine translation models or classification models.

## 2.6 Convolutional Neural Networks

Convolutional Neural Networks (CNN) [34] are popular models, based on feedforward networks, commonly used for image recognition tasks. CNNs are inspired by how receptive field works in our vision. A receptive field is a neuron in a specific region of the retina that is activated by an event. Similarly, CNN neurons behave in the same way, each neuron captures a specific feature of the image, and as a whole the network use all these features to distinguish an image from another.

CNNs are made of a sequence of layers:

- Convolutional layer: it consists of filters used for the convolution operation on the input. It is used to extract features from the input.
- Activation layer: the data coming from the convolutional layer passes through an activation function. Usually $ReLU$ is used, which corresponds to $max(0, x)$.
- Pooling layer: performs a downsampling in order to reduce the dimensionality.
- Fully connected layer: given the feature map it produces the scores for each output class.

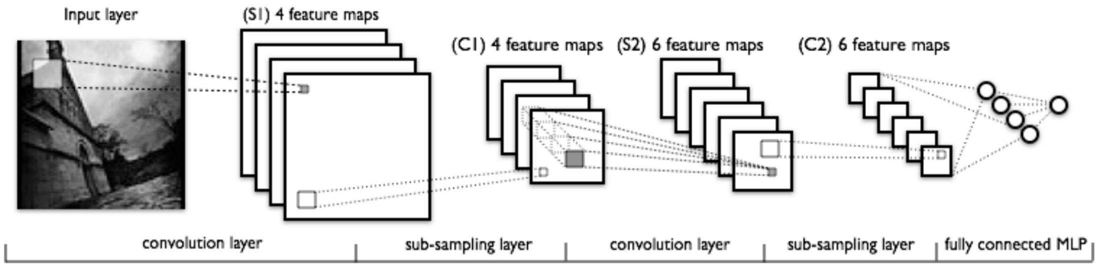


Figure 2.9. Convolutional Neural Network. Source[3]

CNNs are known for their ability to extract features, which make them good for classification tasks. Initially they have been used in image domain but their capabilities allowed them to obtain good results in NLP tasks as well.

### 2.6.1 CNN for Text classification

A successful implementation of CNN for text classification has been introduced by [28]. In this model the input is represented as the concatenation of the words within a sentence. Multiple filters of several dimensions are used for the convolution operation to extract the feature maps. Then a pooling layer is applied over the feature maps to make a downsampling operation. At the end, a fully connected layer is used to generate the probability for each output class.

Figure 2.10 shows the text classification process using a CNN:

- Each word within the sentence is represented with a vector of size $d$ and concatenated following Equation 2.18.

[3]https://skymind.ai/wiki/convolutional-network

- Three filter of size 4, 3, 2 (starting from the top), each type of quantity equals to 2 are used for the convolution operation.
- Each convolution operation gives as result a feature, see Equation 2.19. By translating the filter more features are extracted producing a feature map, see Equation 2.20.
- A 1-max pooling operation $\hat{c} = max\{c\}$ is performed to capture the most important feature within each feature map.
- The processed features are concatenated and a fully connected layer output the probability for each class.

$$x_{1:n} = x_1 \oplus x_2 \oplus \ldots \oplus x_n \tag{2.18}$$

$$c_i = f(W \cdot x_{i:t+h-1} + b) \tag{2.19}$$

$$c = [c_1,\, c_2,\, \ldots,\, c_{n-h+1}] \tag{2.20}$$

### 2.6.2 CNN for Text generation

Apart from classification tasks, CNNs have also been used for language modelling. [29] used CNN in combination with LSTM to represent characters to feed as input. [14] introduced the Gated Convolutional Networks, which takes advantage of the hierarchical architecture of CNNs to extract features over large contexts and use these features to generate text, showing performances comparable to RNNs. [26] and [17] use CNNs to build neural machine translation models that have competitive results compared to the RNN based counterparts.

## 2.7 Word representation

Words in their natural form can not be used as input for the models previously described, since they have no meaning to machines. Therefore, to make them interpretable by the neural network models, they have to be transformed in a numerical representation that identify them. While images and audio have high-dimensional data information, with the common property of being representable in the continuous space, words are considered discrete symbols and need to be handled differently. There are two ways to represent words: on character-level or on word-level. The first method has the advantage of using a small vocabulary, but in addition to learning

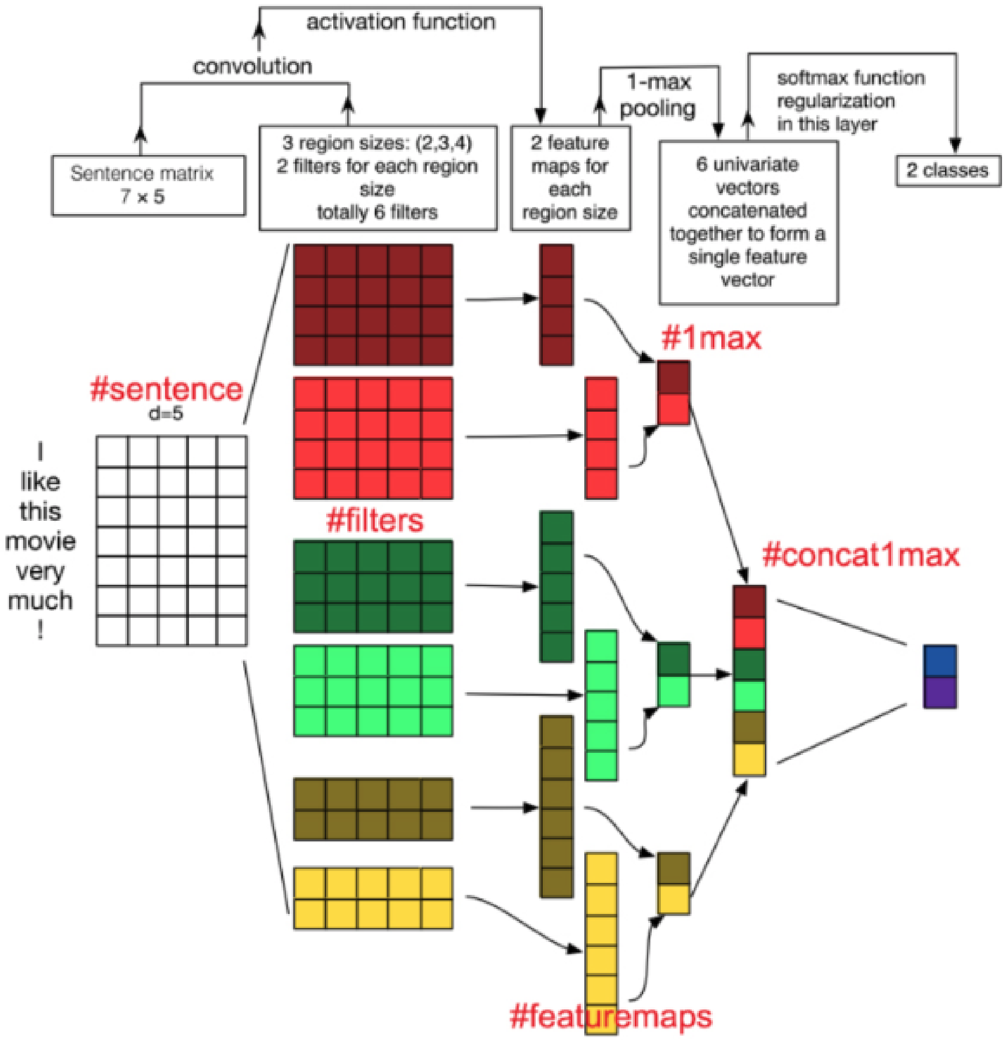


Figure 2.10. Convolutional Neural Network applied in text classication. Source [68]

the sentence structure it needs to learn the spelling. The second method does not have the spelling problem, but in this case the vocabulary is limited.

### 2.7.1 One-hot encoding

A simple method to represent discrete variables is the one-hot encoding technique, in which each word is mapped to a different vector. Given a text, this technique is accomplished by extracting a vocabulary. The vocabulary is a set of unique words in the text. Each word, according to its position in the vocabulary, is associated with a vector. The vector consists of values equal to 0, except one value set to 1, at position $i$, which corresponds to the position of the word within the vocabulary.

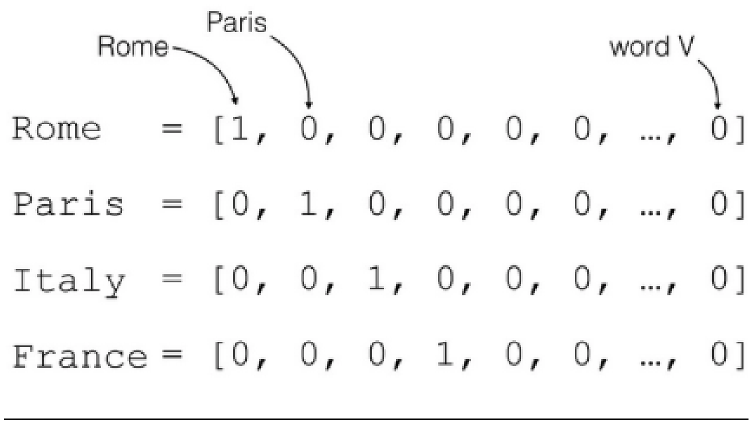

Rome
Paris
word V
Rome = [1, 0, 0, 0, 0, 0, …, 0]
Paris = [0, 1, 0, 0, 0, 0, …, 0]
Italy = [0, 0, 1, 0, 0, 0, …, 0]
France = [0, 0, 0, 1, 0, 0, …, 0]

Figure 2.11. One-hot encoding. Source[4]

One-hot encoding presents two main drawbacks:

- Sparsity problem: this situation occurs when there are a large number of discrete variables to be represented, so the space required is not manageable. For instance, suppose we have $n$ words, it needs to create $n$ vectors of size $n$, so a space complexity of $n^2$, which is very expensive in term of memory.
- Semantic problem: related or similar meaning words are not placed nearby making this method not useful for many NLP tasks.

### 2.7.2 Word Embeddings

To address the previous issues, we use word embeddings, which are continuous vector representation of discrete variables. They provide the properties of dimensionality reduction and semantic representation of words. Dimensionality reduction is provided by making the vector size not dependent on the number of words, so for $n$ words the space required is $n \times m$ where $m$ is the embedding dimension, which is of fixed length. Semantic representation is achieved by mapping words with related or similar meaning close to each other in the vector space. The positions in the space of the word embeddings are learned through machine learning techniques in a supervised environment.

[4]Marco Bonzanini, 2017

Word embedding are employed for several tasks, such as:

- Input of machine learning models: embeddings can be reused, so after they have been trained it is possible to use them in different implementations.
- Finding nearest neighbors in the embedding space: words are placed in the multidimensional vector space based on their relations, this property allows to obtain, given a word, the most similar words.
- Find relations through algebra on the embeddings, for instance if we compute "king − man + woman" the result would be "queen", see Figure [2.12].

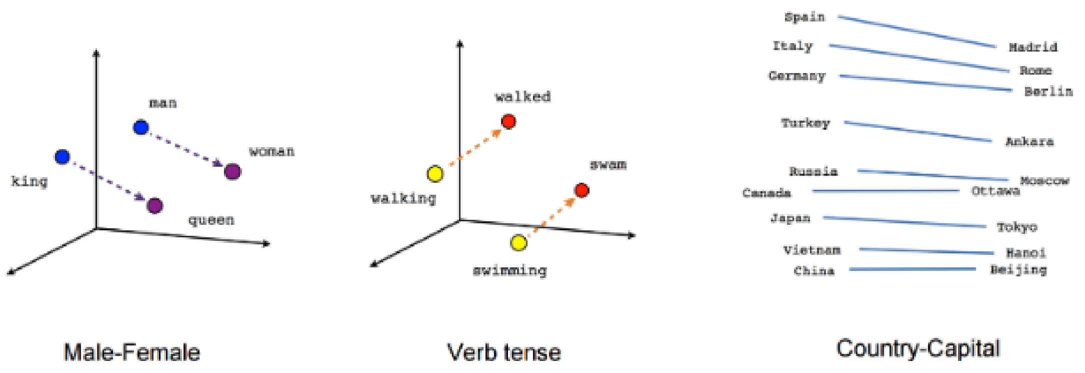


Figure 2.12. Word Embeddings: relationships between words. Source[5]

## Implementations

The simplest method to learn word embeddings is Word2Vec [42], which is a predictive model. The basic idea is based on the assumption that the meaning of a word is determined by the words around it. For each word we compute its embeddings by looking at its surrounding (context words) and determine the value. This process is done in a supervised environment, where the embeddings can be adjusted overtime, in order to improve their representation.

Global vectors (GloVe) introduced by [49] are embeddings create through a count-based method. It is based on the same premises of Word2Vec, so words in the same context have similar meanings. But instead of using a predictive model it computes

[5]https://www.tensorflow.org/tutorials/representation/word2vec

a co-occurrence matrix.

FastText [7] is an extension of Word2Vec, it introduces the concept of n-grams, it considers $n$ characters at a time during the process of creation of the embeddings. The resulting word embedding is the sum of the n-gram embeddings. This method allows to represent words that did not appeared in the training data.

## 2.8 Generative Adversarial Networks

Generative adversarial networks (GAN) are a type of network, of the family of generative models, introduced by [20] and have become very popular for their ability to synthesize images. Given a target data distribution, called real data, GANs objective is to produce data that resembles that distribution. Instead of having an error function that computes the error between the produced data and the expected data like common generative models, GANs take advantage of the capabilities of the models to learn by themselves how likely is the generated data. GANs have two components: a generator and a discriminator. The basic idea is to have these two models competing against each other like a minimax game. The generator tries to produce realistic data to fool the discriminator into thinking they are not fake, whereas the discriminator does its best to identify the fake data.

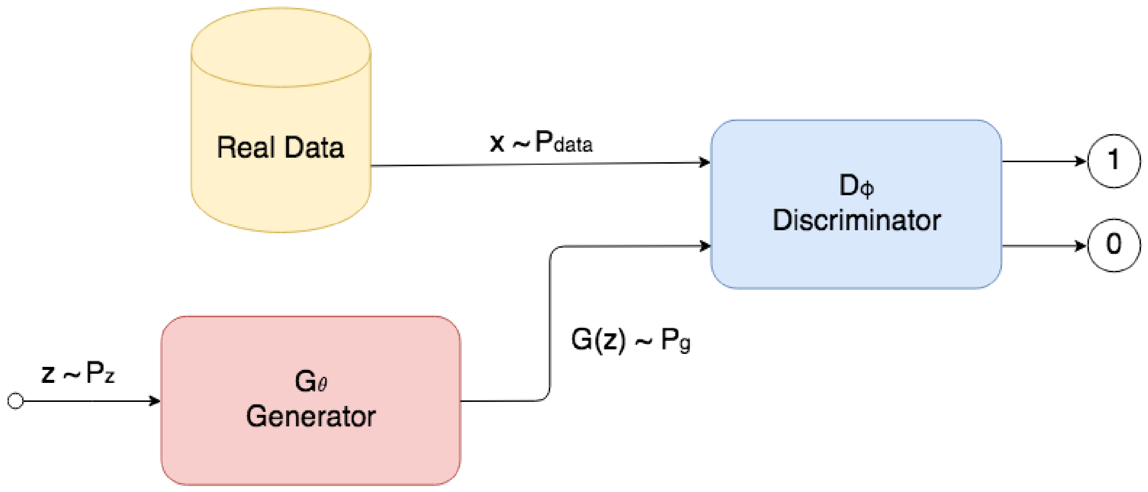


Figure 2.13. Generative Adversarial Network.

### Generator

The generator is a model represented as a function $G$ with differentiable parameters $\theta$. It takes as input a noise $z$ from the distribution $P_z$ and outputs samples $G_\theta(z)$. The sample $G_\theta(z)$ is derived by the generator's distribution $P_g$. The generator's objective is to generate samples from $P_g$ that are close to the distribution of real data $P_{data}$. The samples are then fed to the discriminator and evaluated, a feedback signal is returned to the generator to guide the update process.

### Discriminator

The discriminator is a model represented as a function $D$ with differentiable parameters $\phi$. It takes as input $x$ which is a sample obtained from the generator distribution $P_{data}$ or from the real data. Then it produces a scalar $D_\phi(x)$ as output, for instance the output could be a value between 0 and 1, where 0 denotes the input as fake, whereas 1 as real. Hence, $D_\phi(x)$ represents the probability of the input being from the real data $P_{data}$. Since all the real data instances are labeled as 1 and the fake data are labeled as 0, the discriminator is trained in a supervised environment with cross-entropy as loss function.

### Training

As introduced before, GAN exploits the idea of minimax game between two players based on zero-sum non-cooperative game, in short, if one wins the other loses. Both want to undermine the other, a Nash equilibrium [47] happens when a player will not change its action regardless of what the opponent may do. In other words, it is a state where the action of the opponent does not matter, because it will not change the outcome of the game.

The training process can be summarized as a minimax game with value function $V(D_\phi, G_\theta)$.

$$min_{G_\theta}\ max_{D_\phi}\ V(D_\phi, G_\theta) = E_{x\sim P_{data}(x)}[\log D_\phi(x)] + E_{z\sim P_z(z)}[1 - \log D_\phi(G_\theta(z))] \tag{2.21}$$

$E_{x\sim P_{data}(x)}$ is the expected value of $P_{data}$: the log probability of $D$ predicting that the sample from $P_{data}$ distribution is real.
$E_{z\sim P_z(z)}$ is the expected value of $P_z$: the log probability of $D$ predicting that the sample from $P_z$ distribution is fake.

The two players work against each other, in detail:

- The generator G minimize the probability of making the discriminator recognize the generated data as fake data.

- The discriminator D maximize the probability of correctly distinguish the generated data and the real data.

The optimization of the Equation 2.21 is equivalent to minimize the Jensen-Shannon divergence between the distribution $P_{data}$ and $P_g$ :

$$JS(P_{data}, P_g) = D_{KL}\left(P_{data}\middle|\middle|\frac{P_{data}+P_g}{2}) + D_{KL}(P_g\middle|\middle|\frac{P_{data}+P_g}{2}\right) \tag{2.22}$$

where $D_{KL}$ is the Kullback-Leiber divergence which denotes the difference between two distributions. Therefore the solution for [2.22] is for $P_g$ to be close to the distribution $P_{data}$, in other words the generator have to produce $P_g = P_{data}$.

### Issues

GAN models present some major problems due to being highly susceptible to hyperparameters change and training instability [2], such that an improvement made to a model might cause the worsening of the other one. The convergence happens when the Nash equilibrium is reached and the discriminator cannot distinguish the generated data from the real one anymore. However to reach this state, GANs optimize the Equation [2.22], this means the models keep updating to improve themselves and an update of the discriminator might lead the generator in the opposite direction and vice versa, causing a non-convergence situation. GANs training unbalance is a common problem. As the discriminator accuracy gets higher it becomes more confident in recognizing the fake data. This scenario cause the gradient to vanish [2], therefore the generator in unable to improve. Mode collapse is another issue that can prevent the model to generate useful data. It occurs when the generator maps multiple input $z$ to the same output $x$, causing lack of variation in the generated samples. The reason is that the discriminator feedback has pushed the generator to generate a limited set of sample, with which the generator is able to fool the discriminator.

### Application in Text generation

The main difference between the common generative models and GANs is that the first are directly exposed to the real data with the objective of maximizing the likelihood function that represent that data. Instead in GAN models, the generator has no direct access to the real data, but it is guided by the discriminator, which has access to both the generated and real data. This combination allows GANs to synthesize novel samples.

GANs were originally designed to deal with continuous data, where the discriminator can directly give a feedback signal to the generator via gradient descent, telling how to slightly change the synthetic data to make it more realistic [19]. Since natural language is based on discrete data applying these kind of update is not possible. The main reason is that the sampling process of the sequences based on multinomial are not differentiable, therefore the gradient cannot backpropagate.

An attempt to overcome the differentiability problem has been made by [32], their model exploits the Gumbel-Softmax distribution [25], which is a way to approximate discrete variables as softmax distributions. With this trick is possible to directly backpropagate the gradient from the discriminator to the generator. A different approach to the problem has been proposed by [65] with SeqGAN, see Section [2.9].

## 2.9 SeqGAN

SeqGAN [65] is a model based on GAN for discrete sequence generation. It manages the discrete nature of natural language by using policy gradient [58]. It considers the problem of predicting the next word as a reinforcement learning problem:

- the state is the current generated sequence at the current timestep: $s = (y_1,\, y_2,\, \dots,\, y_{t-1})$
- the action is the next word to be selected: $y_t$
- the policy model applied is based on REINFORCE algorithm [63] and it is denoted by $G_\theta(y_t|\, s)$

### Generator

The generator used is an LSTM model defined as $G$ with parameters $\theta$, this model takes as input the sequence embeddings $\{x_1,\, x_2, \dots, x_n\}$ and produces a sequence of hidden states $\{h_1,\, h_2,\, \dots,\, h_n\}$, this process is denoted as the function $h_t = G_\theta(h_{t-1},\, x_t)$, a $softmax$ layer produces the token probability distribution for the output. The common objective of the generative models in GAN is to maximize the probability of generating a sequence that it is classified by the discriminator as real, in the case of SeqGAN the generator's objective is to generate a sequence to maximize the expected reward, see Equation [2.23].

### Discriminator

The discriminator model used is a CNN classifier denoted as $D$ with parameters $\phi$. It is based on [28] and improved with the highway architecture [55], which can be defined as a gating mechanism that adjusts how the information flows through the network. In simple, it prevents the loss of information in deep neural networks by having a connection from low level layers to high level layers. The discriminator's objective is to minimize the cross-entropy between the ground truth label and the predicted label, in other words it has to correctly classify the real data as real and the generated data as fake.

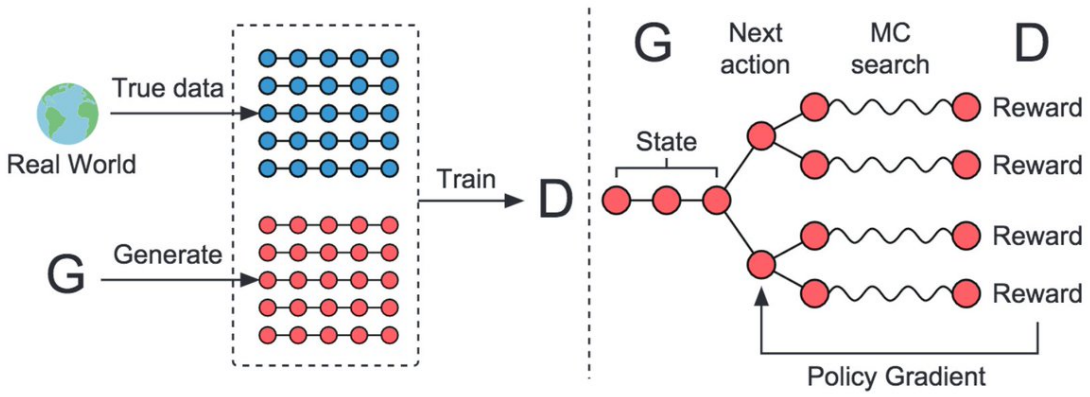


Figure 2.14. SeqGAN architecture: on the left the discriminator training process, on the right the generator training process. Source [65]

## Policy Gradient

SeqGAN uses reinforcement learning to deal with the problem of discrete data. It exploits the policy gradient method REINFORCE to encourage the generator to take the actions that fool the discriminator by maximizing the expected rewards.

$$J(\theta) = E[R_T|\ s_0,\ \theta] = \sum_{y\in\gamma} G_\theta(y_1|\ s_0)\ \cdot Q^{G_\theta}_{D_\phi}(s_0,\ y_1) \tag{2.23}$$

The expected reward $R_T$ can be summarized by the formula [2.23], $y_1$ is the action taken from the state $s_0$, $Q^{G_\theta}_{D_\phi}(s_0,\ y_1)$ is the action-value which is the expected reward given by $D_\phi$ by taking action $y_1$ from state $s_0$ and following policy $G_\theta(y_1|\ s_0)$. $\gamma$ represents the action space, in this case the vocabulary.

Since the rewards are given only when the generations have been completed, in order to evaluate each action, SeqGAN uses Monte Carlo search [8] with a rollout policy. Given a sample $S = \{w_1,\ w_2, \dots,\ w_n\}$, for each word $w_t$, it generates via MC search $n$ samples, the discriminator $D$ evaluates the samples and assign to the word $w_t$ a reward that suggest how much $w_t$ contributed to make the sample classified as real.
The reward given by Monte Carlo search is defined as:

$$Q^{G_\theta}_{D_\phi}(Y_{1:t-1},\ y_t) = \begin{cases} \frac{1}{N}\sum_{n=1}^{N} D_\phi(Y^n_{1:T}) \qquad Y^n_{1:T} \in MC^G(Y_{1:t},\ N) & for\ t < T \\ D_\phi(Y_{1:T}) & for\ t = T \end{cases} \tag{2.24}$$

Where:

$D_\phi(Y^n_{1:T})$: is the probability of the $n$-th sentence being from the real

$Y^n_{1:T}$: is the $n$-th sentence sampled with MC search

$T$: is the length of the sentence sampled with MC search

$N$: is the number of sentences sampled with MC search

Once the reward has been computed, the generator is updated via policy gradient, defined in Equation 2.25. In other words the generator maximize the expected reward Equation 2.23 by minimizing the loss of the generator $\triangledown_\theta G_\theta$ multiplied by the reward $Q^{G_\theta}_{D_\phi}$.

$$\triangledown J_{G_\theta} = \sum_{t=1}^{T} \sum_{y_t \in \gamma} Q^{G_\theta}_{D_\phi}(Y_{1:t-1},\, y_t) \triangledown_\theta G_\theta(y_t |\, Y_{1:t-1}) \tag{2.25}$$

# Chapter 3

# Approach

The goal of this work, as formulated in the introduction, is to use generative models to test their capabilities in the text generation task. In this chapter we describe the models used for our experiments, in particular we build multiple implementations of SeqGAN with different discriminators and compare their performances. Section 3.1 describes the model we have used to define a baseline for our comparisons. Then Section 3.2 introduces the models used in the experiments and Section 3.3 discuss the motivation of our decisions. Section 3.4 explain the training process of a SeqGAN model.

## 3.1 Baseline

As baseline we create a language model based on a RNN. We use the LSTM implementation since it performs better for long term dependencies. The model is represented as $\pi$ with differentiable parameters $\theta$. It is trained using the maximum-likelihood estimation (MLE) method, which consists of optimizing the model parameters in order to fit the observed data. In practice, MLE is computed by minimizing the cross-entropy loss over a given corpus between the predicted output distribution $\hat{y}$ and the expected output $y$ see Equation 3.1, where $T$ is the number of words in the corpus. The cross-entropy loss for each timestep is defined in Equation 3.2, where $|V|$ is the vocabulary, which is the set of unique words within a corpus.

$$J(\theta) = \frac{1}{T} \sum_{t=1}^{T} H(\hat{y}^t, y^t) \tag{3.1}$$

$$H(y^t, \hat{y}^t) = -\sum_{i=1}^{|V|} y_i^t \log \hat{y}_i^t \tag{3.2}$$

Figure 3.1 shows, with an example, the structure of the model. It has an embedding layer that takes as input the words ($w_1$, $w_2$, $\ldots$, $w_n$) of a sequence and outputs the word embeddings ($e_1$, $e_2$, $\ldots$, $e_n$). These are used as input for the LSTM network, which at each timestep $t$ takes as input $e_t$ and $h_{t-1}$ and produce a new hidden state $h_t$. The hidden state is processed by a fully connected layer that produce raw values that are converted in probability distribution by the $softmax$ layer. Lastly, the most likely word is chosen with the $argmax$ function.

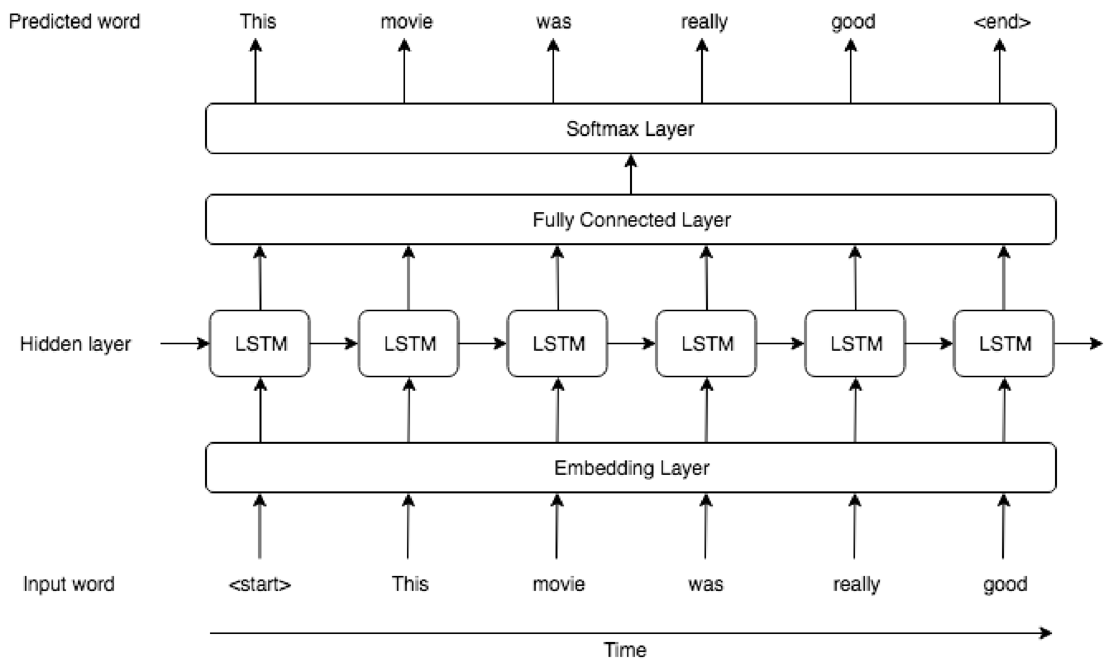


Figure 3.1. RNN language model based on LSTM

## 3.2 Implementation

For the experiments we have developed a framework based on SeqGAN, Figure 3.2. It implements as generator a LSTM network with the same structure showed in the previous Section 3.1. For the discriminator part, we use four different models including the one used in the original implementation, which is a CNN classifier. The other three models are based on RNN and attention mechanism. The goal is to test the ability of the RNN-based classifier to provide meaningful rewards to the generator. Therefore we use the following discriminator implementations:

- Convolutional neural network for text classification, this is the same discriminator implemented in the original SeqGAN described in Section 2.9. Hence, it is based on [38] with the improvements brought by the highway networks [55].

- Attention-based Bidirectional LSTM introduced by [69]. Given that the author showed this model effectiveness when compared to the RNN-based models without attention and CNN-based models, we chose to implement it. The architectural implementation is discussed in Section 3.2.2.

- A structured self-attentive sentence embedding introduced by [39]. We implement this model because it showed good results in a variety of classification tasks compared to other models. This model structure is described in Section 3.2.1.

- Bidirectional IndRNN classifier with attention. This model has the same structure as "Attention-based Bidirectional LSTM" with the only difference that instead of using LSTM implementation it uses IndRNN. The reason we have added this implementation is because the author [36] showed that they work well in classification tasks achieving a low error rate with respect to the counterpart based on LSTM.

The above implementations will be referred as SeqGAN with CNN, SeqGAN with LSTM, SeqGAN with SELF and SeqGAN with IndRNN respectively.

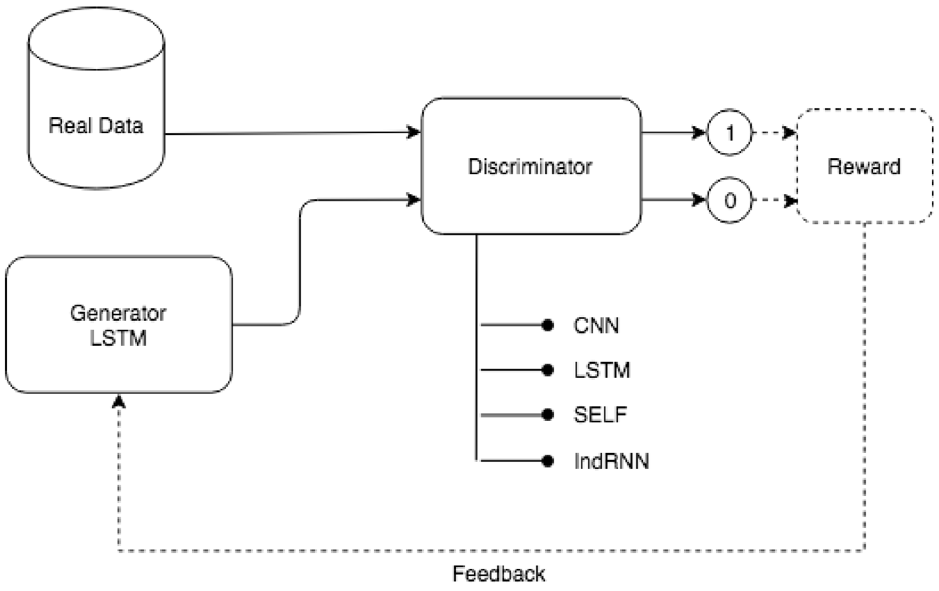


Figure 3.2. Structure of the implementation: LSTM generator and four discriminator implementations.

### 3.2.1 Attention-based Bidirectional LSTM

This classifier model has been proposed by [69]. It is based on [67], which uses a bidirectional RNN model as classifier, but as explained in Section 2.5.1, a vanilla RNN implementation is prone to be affected by the vanishing gradient problem. To solve this issue, this model uses a bidirectional LSTM implementation. The structure can be seen in Figure 3.3.

- An embedding layer transforms the input sentence into word embeddings.
- The LSTM layer takes the word embeddings as input and computes the hidden states $H = \{h_1, \ldots, h_n\}$, where $n$ is the sentence length.
- An attention layer produces the attention weights denoted as $\alpha = softmax(w^T \cdot tanh(H))$, where $w$ refers to a learned matrix weight.
- The context vector is created as $r = H \cdot \alpha^T$.
- The sentence representation is then obtained as $h^* = tanh(r)$.
- A fully connected layer is used to produce the probability distribution for the output classes.

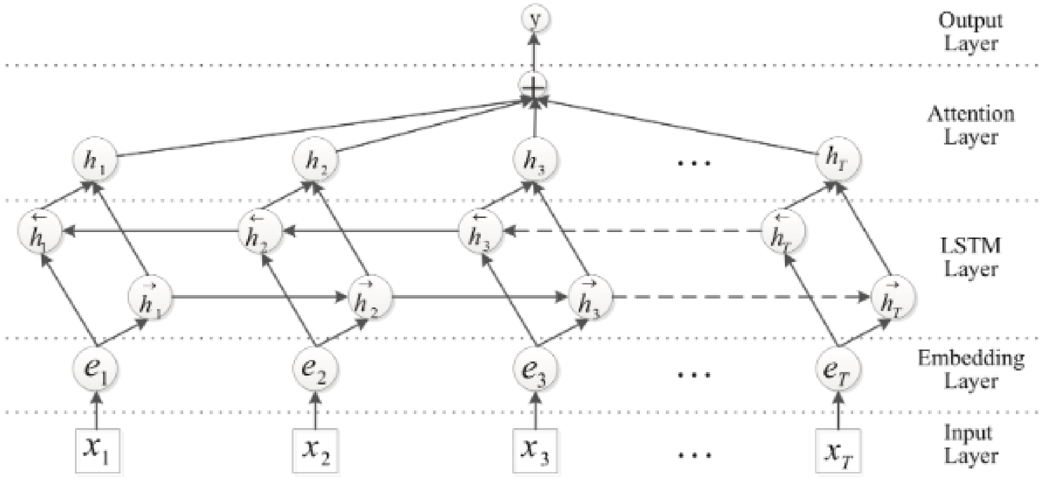

Output Layer
Attention Layer
LSTM Layer
Embedding Layer
Input Layer
y
$h_1$ $h_2$ $h_3$ ... $h_T$
$\overleftarrow{h}_1$ $\overleftarrow{h}_2$ $\overleftarrow{h}_3$ $\overleftarrow{h}_T$
$\overrightarrow{h}_1$ $\overrightarrow{h}_2$ $\overrightarrow{h}_3$ $\overrightarrow{h}_T$
$e_1$ $e_2$ $e_3$ ... $e_T$
$x_1$ $x_2$ $x_3$ ... $x_T$

Figure 3.3. Attention - Based Bidirectional LSTM structure [69]

### 3.2.2 A structured self-attentive sentence embedding

This model proposed by [39] is also based on bidirectional LSTM and attention mechanism. The main difference is how the attention is applied. Figure 3.4 shows the architecture of the model and the structure of the attention mechanism, which the author refers as self-attention. The strength of this model is the ability to represent a variable length sentence into a fixed size representation.

- An embedding layer converts the input sentence into word embeddings and processed by the LSTM network.
- The hidden states are represented as $H = \{h_1, h_2, \ldots, h_n\}$, where $n$ is the sentence length and the size of each hidden state is denoted as $2u$.
- The attention mechanism produces the attention weights as follows: $A = softmax(W_{s2}\ tanh(W_{s1}\ H^T))$, where $W_{s1}$ size is equal to $d_a * 2u$ and $W_{s2}$ size is equal to $r *\ d_a$, $d_a$ is an hyperparameter and $r$ is the number of features to extract.
- Then the sentence representation is obtained as $M = AH$.
- By using a fully connected layer is possible to obtain the probabilities for the output classes, for instance in our case the class 0 represents fake data and class 1 represents real data.

## 3.3 Motivation

The main reason we used GAN is because it has been demonstrated to be effective in text generation. Also its training process allows to overcome the exposure bias problem [65]. This problem [4] occurs to RNN language models due to the inconsistency between the training stage and the inference stage. The first stage, also known as "teacher forcing" [33], expects the training data as input, so it has a direct exposure to the real data. The inference stage, known as "Free Running" [33], use the model prediction at the previous timestep as current input, so an error could cause subsequent errors in later timesteps. GANs are not affected by this problem, because their exposure to the real data is indirect [13]. In other words, the generator training process is not based on explicitly maximizing the likelihood of the real data. Instead they use their own prediction and the feedback received by the discriminator to improve themselves in order to approximate the real data.

The ability of the generator to mimic the real data depends on the feedback of the discriminator, which basically is a text classifier. Its goal is to learn what are the features that define a realistic text. Classifiers based on CNN or RNN lead

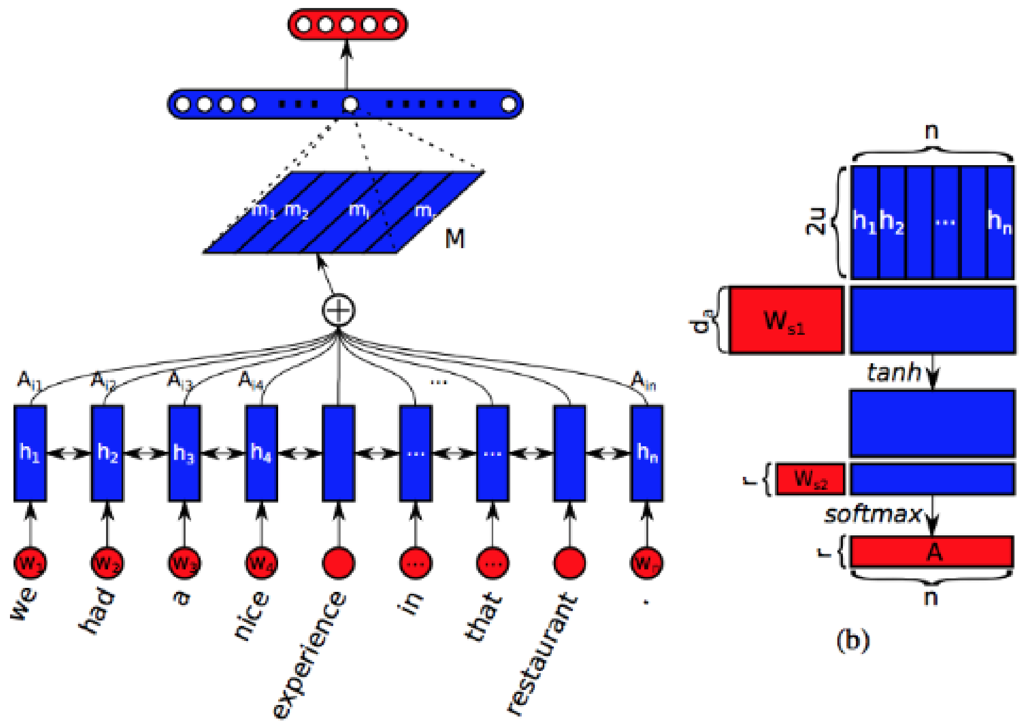


Figure 3.4. Self-attentive sentence embedding structure [39]

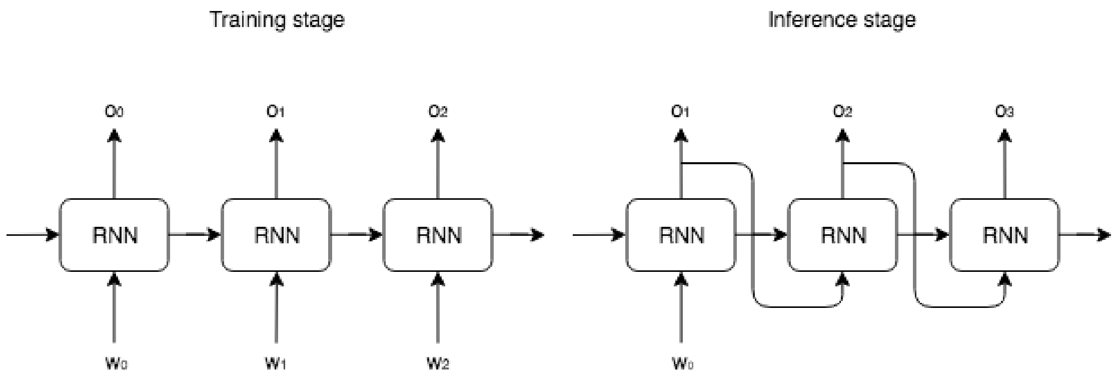


Figure 3.5. Exposure bias problem: on the left the training state known as “Teacher Forcing”, on the right the inference stage known as “Free Running”

to different results in terms of accuracy [64] and amount of information captured, since they operate in a different way. The first model extracts features following a hierarchical schema while the second one computes a representation of the input sequentially.

There are several GAN models implementation based on SeqGAN. RankGAN [37] uses an adversarial ranker as discriminator to provide a better feedback signal to the generator. MaliGAN [11] introduces variance reduction techniques by rescaling the reward and uses a discriminator based on a bidirectional GRU. LeakGAN [21] implements an advanced architecture based on hierarchical reinforcement learning [60] and uses a CNN discriminator to provide information to the generator.

Based on the fact that there are not many implementations based on RNN as discriminator, we tried to use different discriminators based on RNN and attention mechanism to test their effectiveness.

## 3.4 Training process

In this section we describe the training process of the model based on SeqGAN. SeqGAN has two components a generator $G$ with parameters $\theta$ and a discriminator $D$ with parameters $\phi$. Algorithm 1 shows the training process.

**Algorithm 1** SeqGAN training procedure.

```
 1: Initialize $G_\theta$ and $D_\phi$ with random weights $\theta$ and $\phi$
 2: Pre-train $G_\theta$ on real data $S = \{X_{1:T}\}$ using MLE
 3: Generate samples $Z = \{E_{1:T}\}$ using $G_\theta$
 4: Pre-train $D_\phi$ on $\{S : 1, Z : 0\}$ by minimizing the cross-entropy loss
 5: $N$ number of adversarial training epochs
 6: $M$ number of discriminator training epochs
 7: $E$ interval for interleaved training
 8: for i = 1, 2, ..., $N$ do
 9:     Generate a sequence $Y_{1:T}$ using $G_\theta$
10:     Compute the rewards of $Y_{1:T}$ using Monte Carlo search, Equation 2.24
11:     Update generator via policy gradient, Equation 2.25
12:     if i % E == 0 then
13:         Train $G_\theta$ on real data $S = \{X_{1:T}\}$ using MLE
14:     end if
15:     for j = 1, 2, ..., $M$ do
16:         Generate samples $Z = \{E_{1:T}\}$ using $G_\theta$
17:         Train discriminator $D_\phi$ on $\{S : 1, Z : 0\}$
18:     end for
19: end for
```

As [65] points out, SeqGAN requires a pre-training process in order to start the adversarial training. The reasons is that if the generator does not reach a good quality in its generated text, the discriminator can, since the beginning, distinguish the real data from the generated data with high accuracy. This means that the feedback signal emitted by the discriminator is not able to guide the generator in the right direction. The discriminator also has to be pre trained to not be fooled by the generator, otherwise it cannot provide any meaningful feedback signal.

During the adversarial training the generator is trained to maximize the expected reward given by the REINFORCE algorithm 2.23. To achieve this it exploits Monte Carlo search to generate multiple samples to be evaluated by the discriminator. The discriminator produces a reward using Equation 2.24 and the generator use that feedback to update itself via Equation 2.25. After that the discriminator use the real data and generated data labeled as real and fake respectively to improve itself by minimizing the cross-entropy loss between the predicted label and the expected label.

An issue to take into account is when the discriminator provide a wrong update signal, leading the generator to the wrong direction. As suggested by [35], we also test the Teacher Forcing technique [33]. It consists of alternating the adversarial training process with the MLE training using the real data. So in case the generator has received a misleading update it is redirected to the right path. A similar suggestion has been made by [21], which refers this technique as "interleaved training", in this case after a predefined number of adversarial epochs follows a predefined number of MLE training epochs.

# Chapter 4

# Experimental Setup

In this chapter we provide the information about the tools we have used for our experiments. Section 4.1 lists the framework and libraries used to implement our models and metrics. Section 4.2 describes the datasets used and the pre-processing applied to them. Then Section 4.3 shows the metrics used for the evaluation.

## 4.1 Implementation

For the experiments we used PyTorch (0.4.1)[1], which is a popular deep learning framework developed by Facebook. It allows to create models using high level API and also it provides low level API for custom implementations. Metrics are implemented with the help of Natural Language Toolkit (NLTK) library based on Python.

## 4.2 Datasets

For our experiments we use three datasets: Image COCO [38], EMNLP News [2] and Amazon Reviews [3]. We test the models with different dataset, each one with its own characteristics. The first dataset is used to test the ability to generate short text, the second is used to test medium length generation, while the last is for testing the generation of long sentences.

[1]https://www.pytorch.org
[2]https://github.com/geek-ai/Texygen/tree/master/datal
[3]https://snap.stanford.edu/data/web-FineFoods.html

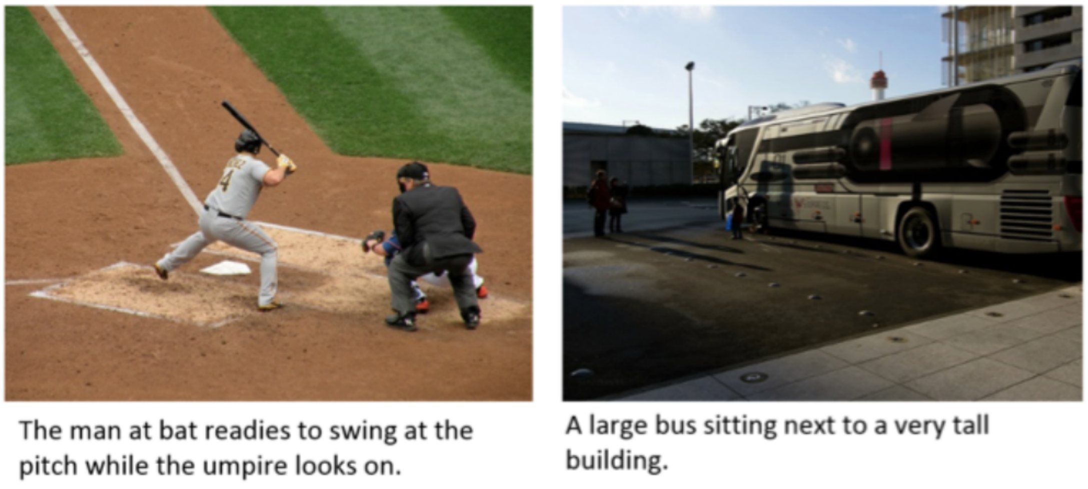
### 4.2.1 Image COCO

Image COCO [38] is a dataset developed by Microsoft used for object detection and captioning. The dataset consists of annotated images, see Figure 4.1, such that each image has a description of the objects illustrated or the actions performed by the subjects in the image. Since we need only to work on text, we use only the annotations.

For this experiment we extract a sample made of 10,000 sentences for the training set and other 10,000 sentences as test set for the evaluation. The captions are converted in lowercase and we divide the words from the punctuation (e.g. "A bathroom with a toilet, sink, and shower." → "a bathroom with a toilet , sink , and shower . " ), so that the models can learn how to appropriately use punctuation.

Since the sentences are of variable length, we apply special tokens to define the start and the end of the sentence, we also add a padding token to uniform all the sentences length because it is a requirement for them to be processed in parallel by the generative models, see Table 4.1.

For this dataset we use the pre-trained GloVe embeddings [49] because they cover the majority of the vocabulary. These embeddings have been created with a training on Wikipedia articles and on Gigaword 5 archive, for a total of 6 billion tokens and 400,000 distinct words.



The man at bat readies to swing at the pitch while the umpire looks on.

A large bus sitting next to a very tall building.

Figure 4.1. Image COCO example [38]

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | A | man | enjoys | cooking | food | in | a | pan. | | |
| | A | man | and | a | dog. | | | | | |
| <sos> | a | man | enjoys | cooking | food | in | a | pan | . | <eos> |
| <sos> | a | man | and | a | dog | . | <eos> | <pad> | <pad> | <pad> |

Table 4.1: Example of data pre-processing.

| **Image COCO dataset** | **Train set** | **Test set** |
|---|---|---|
| Number of sentences | 10,000 | 10,000 |
| Number of tokens | 113,790 | 112,108 |
| Number of words | 4,642 | 4,237 |
| Sentence average length | 11.38 | 11.21 |
| Sentence max length | 25 | 34 |
| Sentence min length | 7 | 7 |
| Max word count | 16,977 | 16,177 |
| Min word count | 1 | 1 |
| Word count equal to 1 | 2,066 | 1,896 |
| GloVe 6B coverage | 95.58% | 95.93% |

Table 4.2: Image COCO dataset statistics: number of words refers to the number of unique words and tokens refers to the total number of words

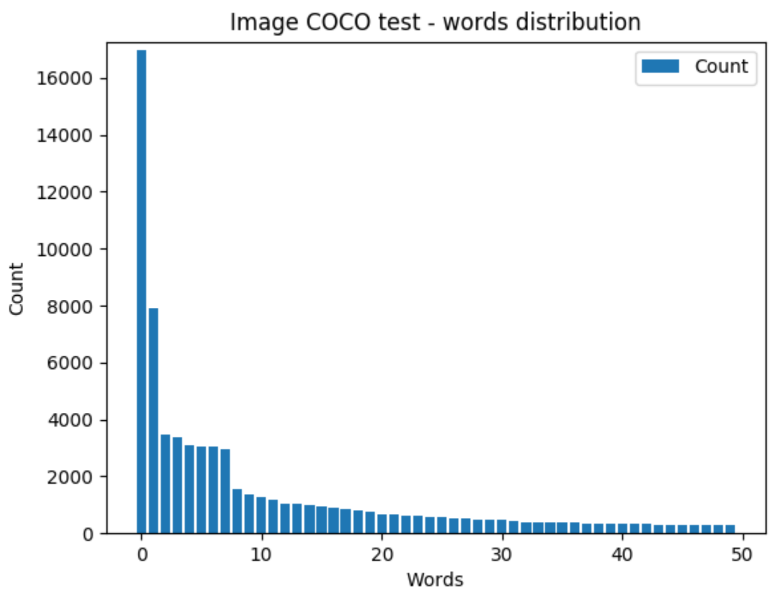


Figure 4.2. Image COCO words distribution: it shows the number of occurrence of the first 100 words.

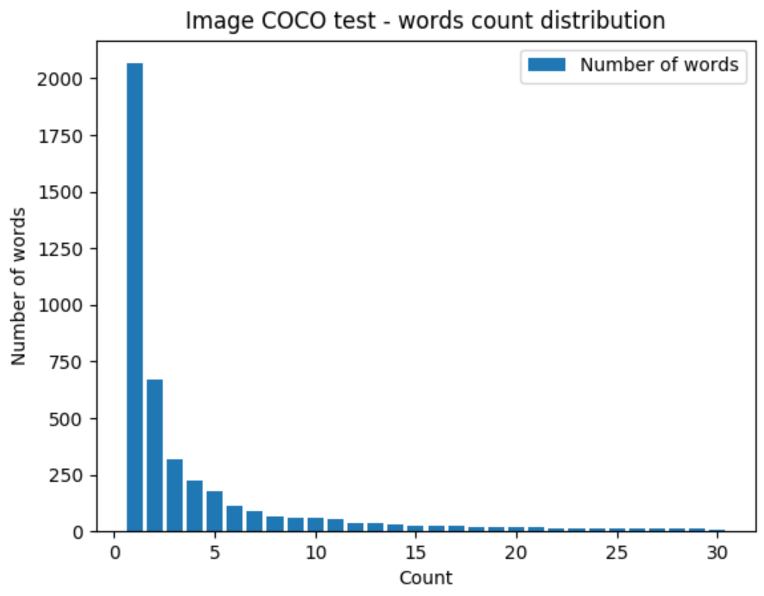


Figure 4.3. Image COCO words count distribution: it shows the how many words per count, e.g. there are about 2000 words which occurrence is equal to 1.

### 4.2.2 EMNLP News

EMNLP news 2017 is a dataset that contains sentences of medium length. Since the dataset has 278,586 rows we extract a sample of 10,240 rows having maximum length 25. For this dataset we do not perform a lower case processing because it contains proper nouns and we want to preserve the meaning of those words. The punctuation is divided as before and the special tokens are applied to uniform the sentences. In this case we use learnable word embeddings since GloVe does not offer good coverage.

| EMNLP News | Train set | Test set |
|---|---|---|
| Number of sentences | 10,240 | 10,000 |
| Number of tokens | 222,197 | 278,141 |
| Number of words | 5,119 | 5,134 |
| Sentence average length | 21.7 | 27.8 |
| Sentence max length | 24 | 49 |
| Sentence min length | 21 | 21 |
| Max word count | 10,321 | 12,769 |
| Min word count | 1 | 1 |

Table 4.3: EMNLP News dataset statistics.

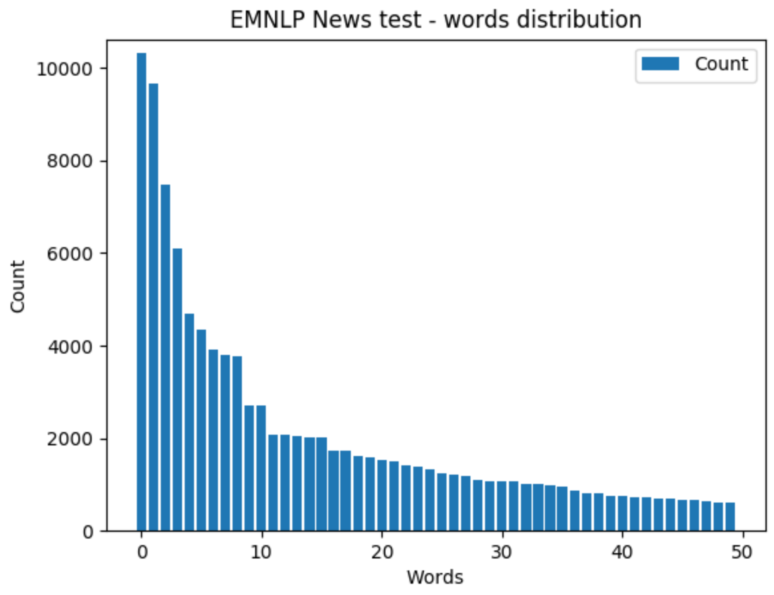


Figure 4.4. EMNLP News words distribution: it shows the number of occurrence of the first 100 words.

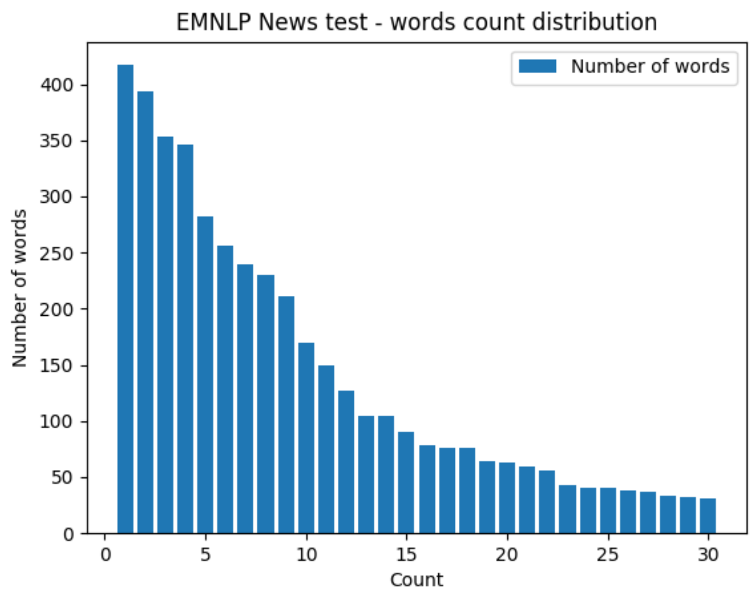


Figure 4.5. EMNLP News words count distribution: it shows the how many words per count, e.g. there are about 400 words which occurrence is equal to 1.

### 4.2.3 Amazon Reviews

This dataset contains reviews of fine foods from Amazon. We parse the dataset and discard all the products information but the text reviews. We leave the upper case and lower case as it is. We perform the punctuation pre-processing and apply the special tokens. Also in this case we use learnable embeddings because GloVe does not offer good coverage.

| **Amazon Reviews** | **Train set** | **Test set** |
|---|---|---|
| Number of sentences | 10,240 | 10,240 |
| Number of tokens | 302,899 | 302,485 |
| Number of words | 12,880 | 12,609 |
| Sentence average length | 29.57 | 29.53 |
| Sentence max length | 48 | 50 |
| Sentence min length | 24 | 12 |
| Max word count | 13,721 | 13,710 |
| Min word count | 1 | 1 |

Table 4.4: Amazon Reviews dataset statistics.

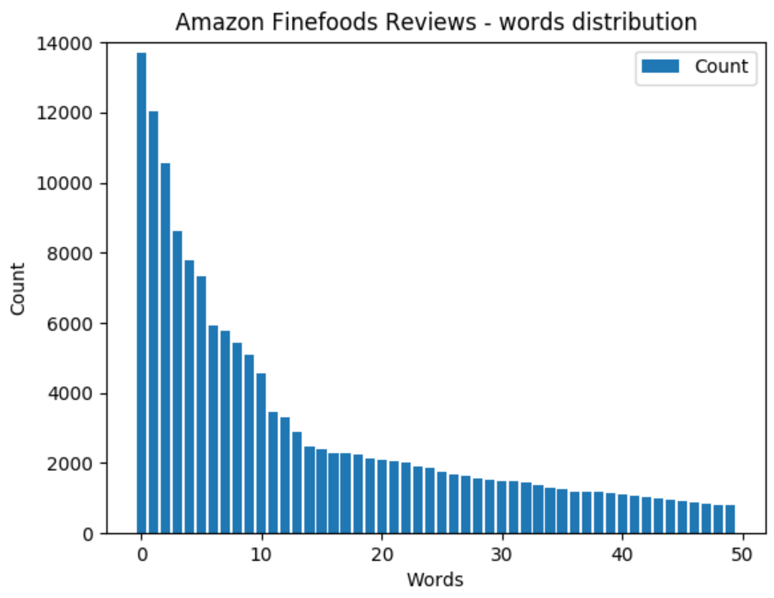


Figure 4.6. Amazon Reviews words distribution: it shows the number of occurrence of the first 100 words.

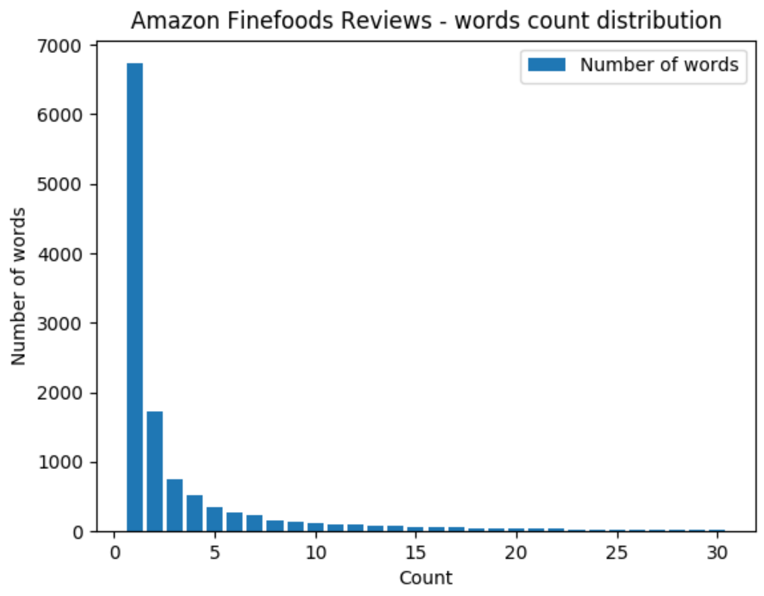


Figure 4.7. Amazon Reviews words count distribution: it shows the how many words per count, e.g. there are about 6600 words which occurrence is equal to 1.

## 4.3 Metrics

The evaluation of generative models is not a trivial-task. The current available metrics are based on word-overlapping methods, word embedding and readability analysis. Readability is defined as the effort needed for a reader to understand a written text. The complexity of a text depends on multiple factors such as the vocabulary used and the syntax structure. The objective of a written text is to be able to communicate the content in order to engage the reader. Since these metrics do not consider all the aspects of the natural language, they still suffer from not having a strong correlation with human judgment. In this work we use a set of metrics that measure different aspects of the generated text:

- BLEU
- POS-BLEU
- SELF-BLEU
- Flesch Reading Ease
- Coleman-Liau
- Gunning Fog Index

### 4.3.1 BLEU

Bilingual Evaluation Understudy-[48, 50, 65] is a metric introduced with the intent of evaluating the performances of the machine translation systems. This metric requires a human quality corpus used as reference for comparison to produce a numerical value that suggest the translation closeness. BLEU is based on word-overlapping and uses weighted geometric mean, see Equation 4.3, multiplied by a brevity penalty, see Equation 4.2 to combine the n-grams precision, see Equation 4.1, between the candidate translation and the reference translation. N-gram precision is obtained by counting the number of n-gram matches in relation to the total number of n-gram within the reference. Brevity penalty is used as a counterweight to avoid giving high scores to short candidates. The metric outputs a value between 0 and 1, a value close to 1 means a higher number of n-grams are matching the reference, so the quality is better.

$$p_n = \frac{\sum_{n \in cand} count_{matched}(n)}{\sum_{n \in cand} count(n)} \tag{4.1}$$

$$BP = min\big(1, \frac{candidate\ length}{reference\ length}\big) \tag{4.2}$$

$$BLEU\text{−}n = BP\ \Big(\prod_{i=1}^{n} p_i\Big)^{\frac{1}{n}} \qquad for\ i = 1,\ \ldots,\ n \tag{4.3}$$

Where:

- $n$ is the n-gram target.
- $cand$ is the candidate phrase.
- $p_n$ is the precision for n-gram.
- $BP$ is the brevity penalty.

**Example: BLEU score**

Reference: ‘The’, ‘quick’, ‘brown’, ‘fox’
Candidate: ‘The’, ‘fast’, ‘brown’, ‘fox’
n-grams: 4

precision-1 = 3 / 4 → (‘The’) (‘brown’) (‘fox’)
precision-2 = 1 / 3 → (‘brown’, ‘fox’)
precision-3 = 0
precision-4 = 0
BP = 1
BLEU-4 = 0.707

### 4.3.2 POS-BLEU

POS-BLEU [50], part-of-speech BLEU, computes the BLEU score, which ranges between 0 and 1, using part-of-speech (POS) tags instead of the words. It takes into account the syntax aspects of a sentence. It is used to measure the structure similarity between the candidate phrase and the reference phrase. It first runs a part-of-speech tagging on both the reference and candidate text, then it uses the BLEU metric to evaluate their similarity.

**Example: POS-BLEU score**

Candidate: 'He' 'is' 'going' 'on' 'holiday'
Part-of-speech candidate:
- 'He' → 'Pronoun'
- 'is' → 'Verb'
- 'going' → 'Verb'
- 'on' → 'Preposition'
- 'holiday' → 'Noun'

Reference: 'She' 'was' 'eating' 'pizza'
Part-of-speech reference:
- 'She' → 'Pronoun'
- 'was' → 'Verb'
- 'eating' → 'Verb'
- 'pizza' → 'Noun'

n-grams: 4

precision-1 = 4 / 5
precition-2 = 2 / 4
precision-3 = 1 / 3
precision-4 = 0
BP = 1
POS-BLEU-4 = 0.604

### 4.3.3 SELF-BLEU

SELF-BLEU is a metric proposed in [70], it is used to measure the diversity of the generated text with respect to itself. The main reason is that GANs tend to be affected by the mode collapse problem, which cause the models to generate a limited set of outputs. The score is computed in the same way as BLEU, therefore it has the same range of values, but with the difference that it uses the generated text as reference. So each sentence of the generated text is used as candidate against the whole generated text. A higher score means that the samples from the model often repeat themselves in some part, while a lower score means the model tends to generate different words.

### 4.3.4 Flesch Reading Ease

Flesch Reading Ease [51] is a readability test introduced in 1948. It gives a score between 1 and 100 to suggest which level of education is needed to be able to understand a text. A higher value means that a lower level of education is necessary.

$$206.835 - 1.015\left(\frac{total\ words}{total\ sentence}\right) - 84.6\left(\frac{total\ syllables}{total\ words}\right) \tag{4.4}$$

### 4.3.5 Coleman-Liau

Also this metric is used to compute the grade level necessary to read a text. Unlike the other metrics it considers characters instead of syllables.

$$CLI = 0.05888L - 0.269S - 15.8 \tag{4.5}$$

$L$ represents the average number of letters per 100 word, $S$ represents the average number of sentences per 100 word. The value of CLI ranges between 6 and 17, suggesting the education level required. For instance a value of 8 means the text is for .

### 4.3.6 Gunning Fog Index

This readability test is used to evaluate text of about 100 words, so not for entire text but paragraphs. It takes into account the number of complex words. A word is complex if it consists of three or more syllables and it is not a proper noun. It outputs a value between 6 and 17 as Coleman-Liau index.

$$0.4\left[\left(\frac{words}{sentences}\right) + 100\left(\frac{complex\ words}{words}\right)\right] \tag{4.6}$$

# Chapter 5

# Results

In this chapter we present the results of our experiments. We use the datasets and metrics described in the previous section. Section 5.1 describes the results using the dataset Image COCO, which consists of short sentences. Section 5.2 contains the results for EMNLP News dataset, in this case we evaluate the ability to generate more complex sentences of medium length. Section 5.3 presents the results of the Amazon Reviews dataset, which is made of long sentences. For each section, we first describe the model setting we have used for the experiment, then we show the results obtained from the metrics, after that we comment the quality of some samples generated by the models. In Section 5.4 we discuss and make some consideration about the results.

## 5.1 Test Image COCO

This first experiment tests the capability of generating short text sequences using the annotations of Image COCO dataset.

We use pre-trainined GloVe of size 100 as embeddings for both the generator and discriminators, this gives a slight improvement thanks to the semantic information that GloVe embeddings provide. The generator and discriminator hidden layers dimension are set to 100. The maximum length for the generation is capped to 20, because as shown in Table 4.2, the sentences within the dataset have an average length of less than 20. Monte Carlo search roll-out policy, as explained in Section 2.9, consists of sampling multiple times sub-sequences of a sentence, this process is particularly time consuming. For instance, consider a sentence of length $l$ for $x$ times, it would require $l \times x$ sampling operations to get the reward. Therefore, we set Monte Carlo search rollout policy to 16 samples, which is a good trade-off between the produced results and execution time. All the training processes use Adam optimizer algorithm [30]. The generator pre training process consists of 60 epochs and batch size of 64. The discriminator pre training process consists of 5 epochs, this choice has been made because for more training epochs the discriminator becomes to accurate in distinguishing the data. The adversarial training runs for 100 epochs.

The results are obtained, after the training process has completed, by randomly sampling 300 sentences for 3 times and averaging the scores. Table 5.1(a), 5.1(b), 5.1(c) show the BLEU, POS-BLEU, SELF-BLEU score respectively, for the pre trained model, the RNN baseline and the SeqGAN implementations. Table 5.1(d) presents the readability metrics that put into comparison the test data and the generated data. Figure 5.1(a) reports the loss of the generator and Figure 5.1(b) shows the loss of the discriminator during the adversarial training.

### 5.1.1 Image COCO samples

This section shows some random samples generated by the models. As shown, the models managed to generate sentences with good syntax structure and reasonable coherence. The models learned how to link entities and actions in a meaningful way. Also the contexts within the sentences have been respected in most of the cases, although there are some bizarre mixes that require some interpretation. For instance, some good results are shown with the association of "airplane" with "blue sky" and the action of "sitting" with "in a white bathroom sink".

#### SeqGAN with CNN

- a street full of motorcycles and people walking with their surf boards .
- a counter with wooden floor and a flat screen tv .
- a bathroom with white tile and white walls .

#### SeqGAN with SELF

- an airplane flies overhead in a clear blue sky .
- two people sitting on a desk with a motorcycle by it .
- a toilet sitting next to a brick wall with a green wall .

#### SeqGAN with LSTM

- a picture of a boat full of rowers .
- a man riding a motorcycle next to a red motorcycle .
- a bathroom with a sink and mirror with a black seat .

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| BLEU-2 | 0.7464 | 0.7481 | 0.7837 | 0.7850 | 0.7749 | **0.8077** |
| BLEU-3 | 0.5122 | 0.5319 | 0.5380 | 0.5682 | 0.5306 | **0.5990** |
| BLEU-4 | 0.3202 | 0.3423 | 0.3401 | 0.3641 | 0.3334 | **0.3927** |
| BLEU-5 | 0.1945 | 0.2166 | 0.2120 | 0.2239 | 0.2098 | **0.2456** |

(a) BLEU score - Image COCO: comparison of the BLEU score for 2-gram to 5-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| POS-BLEU-2 | **0.9995** | 0.9974 | 0.9970 | 0.9988 | 0.9987 | 0.9968 |
| POS-BLEU-3 | 0.9950 | 0.9934 | 0.9925 | **0.9957** | 0.9941 | 0.9936 |
| POS-BLEU-4 | 0.9812 | 0.9808 | 0.9816 | 0.9826 | 0.9806 | **0.9840** |
| POS-BLEU-5 | 0.9550 | 0.9495 | 0.9583 | 0.9587 | 0.9554 | **0.9645** |
| POS-BLEU-6 | 0.9095 | 0.8959 | 0.9174 | 0.9230 | 0.9119 | **0.9284** |
| POS-BLEU-7 | 0.8394 | 0.8229 | 0.8547 | 0.8659 | 0.8475 | **0.8742** |

(b) POS-BLEU score - Image COCO: comparison of the POS-BLEU score for 2-gram to 7-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| SELF-BLEU-2 | **0.6745** | 0.6782 | 0.7636 | 0.7259 | 0.7498 | 0.7602 |
| SELF-BLEU-3 | **0.4501** | 0.4567 | 0.5771 | 0.5117 | 0.5536 | 0.5693 |
| SELF-BLEU-4 | **0.2968** | 0.2898 | 0.4232 | 0.3361 | 0.3955 | 0.3994 |
| SELF-BLEU-5 | **0.1983** | 0.1884 | 0.3090 | 0.2160 | 0.2866 | 0.2791 |

(c) SELF-BLEU score - Image COCO: comparison of the SELF-BLEU score for 2-gram to 5-gram

| | Real Data | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| Flesch Reading Ease | 75.74 | 77.87 | 80.06 | 79.37 | 78.18 | 79.32 |
| Coleman-Liau | 6.38 | 5.34 | 4.58 | 5.15 | 5.12 | 4.16 |
| Gunning Fog Index | 11.30 | 10.91 | 9.84 | 10.13 | 10.84 | 10.25 |

(d) Image COCO - Readability metrics

Table 5.1: Metrics results - Image COCO

#### SeqGAN with IndRNN

- a person riding a dirt bike on a city street .
- a bunch of bananas hanging on a small table .
- a small kitten sitting in a white bathroom sink .

## 5.2 Test EMNLP News

This dataset is made of sentences that have a more complex structure compared to Image COCO, this is due to the vocabulary used, which contains proper nouns and several types of punctuation. Also, as shown in Table 4.3, the sentences have more word diversity, in the sense that, unlike Image COCO where few words have high occurrence, EMNLP News has a few more words with high occurrence, we assume that this makes it harder for the model to predict the next word with high confidence.

In this case we use learnable embeddings of size 400. The generator hidden layer size is set to 400, while the discriminator hidden size is set to 100. Monte Carlo search rollout policy is set 16 samples with maximum length of 25. The pre training epochs are set to 150 and 1 respectively for the generator and discriminator. The other parameters are left as in the previous experiment.

Table 5.2(a), 5.2(b), 5.2(c) provide the results of the word-overlapping metrics for each implementation. Table 5.2(d) shows the readability metrics. Figure 5.2(a) and 5.2(b) report the generator and discriminator loss respectively.

### 5.2.1 EMNLP News samples

This section shows some random samples generated by the models about EMNLP News. In this case, we immediately notice that the sentences are not fluent. These sentences are hard to interpret because they lack of global coherence as we often see that they start with a topic and then change the course of the sentence to an unrelated topic. Also they feel incomplete, since by changing the topic, the sentence is meaningless.

#### SeqGAN with CNN

- The team continued to spend a weekend after the start of a public have two places to say .
- By 2001 , there ’ s also has more than 100 million in each year , and the employment market .

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| BLEU-2 | 0.7770 | 0.7834 | **0.8370** | 0.8300 | 0.7994 | 0.8071 |
| BLEU-3 | 0.5231 | 0.5265 | 0.5947 | **0.6002** | 0.5742 | 0.5664 |
| BLEU-4 | 0.2921 | 0.2911 | 0.3489 | **0.3731** | 0.3565 | 0.3342 |
| BLEU-5 | 0.1599 | 0.1620 | 0.1917 | **0.2225** | 0.2187 | 0.1894 |

(a) BLEU score EMNLP News: comparison of the BLEU score for 2-gram to 5-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| POS-BLEU-2 | 0.9116 | 0.9228 | **0.9495** | 0.9414 | 0.9044 | 0.9158 |
| POS-BLEU-3 | 0.9047 | 0.9160 | **0.9420** | 0.9348 | 0.8985 | 0.9086 |
| POS-BLEU-4 | 0.8718 | 0.8804 | **0.9110** | 0.9041 | 0.8709 | 0.8758 |
| POS-BLEU-5 | 0.7906 | 0.7971 | **0.8365** | 0.8340 | 0.8016 | 0.7995 |
| POS-BLEU-6 | 0.6602 | 0.6658 | 0.7105 | **0.7139** | 0.6798 | 0.6715 |
| POS-BLEU-7 | 0.4913 | 0.4943 | 0.5468 | **0.5482** | 0.5237 | 0.5051 |

(b) POS-BLEU score EMNLP News: comparison of the POS-BLEU score for 2-gram to 7-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| SELF-BLEU-2 | 0.5290 | **0.5155** | 0.6371 | 0.6442 | 0.6303 | 0.5972 |
| SELF-BLEU-3 | 0.2641 | **0.2443** | 0.3803 | 0.4139 | 0.4010 | 0.3511 |
| SELF-BLEU-4 | 0.1413 | **0.1229** | 0.2130 | 0.2637 | 0.2665 | 0.2062 |
| SELF-BLEU-5 | 0.0847 | **0.0727** | 0.1279 | 0.1678 | 0.1840 | 0.1232 |

(c) SELF-BLEU score EMNLP News: comparison of the SELF-BLEU score for 2-gram to 5-gram

| | Real Data | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| Flesch Reading Ease | 61.07 | 64.41 | 66.65 | 67.03 | 67.22 | 66.75 |
| Coleman-Liau | 8.47 | 7.32 | 6.16 | 5.90 | 6.49 | 6.26 |
| Gunning Fog Index | 15.84 | 14.76 | 14.38 | 13.85 | 13.77 | 13.87 |

(d) EMNLP News - Readability metrics

Table 5.2: Metrics results - EMNLP News

- The deal were was a huge change they are going to get out of their open - plan for a product of any EU asylum .

#### SeqGAN with SELF

- The data being offered to go through for another two hours to play that more than you ' ve asked how they were
- That ' s how to improve the court charged today , I have to think of this process .
- The other thing now is how to improve , otherwise didn ' t get a big win over Christmas cost one .

#### SeqGAN with LSTM

- I ' m not saying the EU , just two goals and so many other players that the great exist .
- If you need to pay the rent of what he just can win his name , you ' re going to take them with their
- At the time of 2015 , the project could be on track to change .

#### SeqGAN with IndRNN

- While I can start again , and I ' m sorry and she may give up to give you about an effective message .
- I ' m just here because there is a lot to be seen over - that ' s the attention of Mr Adams .
- By the end of which you ' re living as great as conditions in the UK for that newspaper article .

## 5.3 Test Amazon Reviews

This experiment tests the capability of generating long sentences using Amazon Reviews. This dataset has a larger vocabulary than the previous datasets, see Table [4.4].

We use the same configuration as in the EMNLP News experiment, with the only difference that Monte Carlo search rollout has a maximum length of 35. The results are displayed in Table 5.3(a) 5.3(b) 5.3(c) 5.3(d). Figure 5.3(a) shows the generator loss and Figure 5.3(b) display the discriminator loss.

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| BLEU-2 | 0.8804 | 0.8676 | 0.8904 | 0.8953 | **0.9064** | 0.9003 |
| BLEU-3 | 0.6791 | 0.6551 | 0.6860 | 0.7001 | **0.7282** | 0.7174 |
| BLEU-4 | 0.4481 | 0.4157 | 0.4437 | 0.4719 | **0.5040** | 0.4850 |
| BLEU-5 | 0.2598 | 0.2328 | 0.2456 | 0.2792 | **0.2991** | 0.2827 |

(a) BLEU score - Amazon Reviews: comparison of the BLEU score for 2-gram to 5-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| POS-BLEU-2 | **0.9996** | 0.9982 | 0.9992 | 0.9994 | 0.9995 | 0.9969 |
| POS-BLEU-3 | **0.9955** | 0.9943 | 0.9948 | 0.9944 | 0.9944 | 0.9922 |
| POS-BLEU-4 | **0.9723** | 0.9720 | 0.9700 | 0.9701 | 0.9716 | 0.9709 |
| POS-BLEU-5 | 0.9105 | 0.9087 | 0.9043 | 0.9071 | 0.9139 | **0.9160** |
| POS-BLEU-6 | 0.7957 | 0.7971 | 0.7885 | 0.7954 | 0.8105 | **0.8152** |
| POS-BLEU-7 | 0.6273 | 0.6421 | 0.6266 | 0.6388 | 0.6574 | **0.6682** |

(b) POS-BLEU score - Amazon Review: comparison of the POS-BLEU score for 2-gram to 7-gram

| | RNN pre-train | RNN | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| SELF-BLEU-2 | 0.6544 | **0.6281** | 0.7116 | 0.6884 | 0.7346 | 0.7147 |
| SELF-BLEU-3 | 0.3946 | **0.3497** | 0.4790 | 0.4416 | 0.5200 | 0.4919 |
| SELF-BLEU-4 | 0.2013 | **0.1736** | 0.2909 | 0.2444 | 0.3418 | 0.3024 |
| SELF-BLEU-5 | 0.6273 | **0.0967** | 0.1712 | 0.1345 | 0.2103 | 0.1776 |

(c) SELF-BLEU score - Amazon Reviews: comparison of the POS-BLEU score for 2-gram to 5-gram

| | Real Data | RNN MLE | SeqGAN with CNN | SeqGAN with SELF | SeqGAN with LSTM | SeqGAN with IndRNN |
|---|---|---|---|---|---|---|
| Flesch Reading Ease | 65.19 | 75.44 | 80.41 | 78.81 | 81.41 | 76.76 |
| Coleman-Liau | 7.82 | 6.37 | 5.25 | 5.42 | 4.08 | 5.26 |
| Gunning Fog Index | 15.30 | 12.58 | 11.27 | 12.15 | 10.94 | 12.72 |

(d) Amazon Reviews - Readability metrics

### 5.3.1 Amazon Reviews samples

In this section we present the samples generated from Amazon Reviews. Since the dataset contains the reviews from the customers, the language used is pretty informal so the vocabulary used and the sentences structure are less complex than the previous dataset EMNLP News, we can also see this in Table 4.4, where the words distribution is less uniform with few words used very often. As shown, each sample, in most of the cases, is made of multiple sentences. Taken singularly they could have a proper meaning, but if considered as a whole, they sometimes contradict each other. For instance, in the first sentence generated by SeqGAN with CNN, we can see that the first part gives a positive sentiment and suddenly after, it follows a negative comment. Therefore these sentences are hard to make a sense out of them.

#### SeqGAN with CNN

- My dogs love it . It's disgusting . They are expensive , but it's more
- These chips were very good and a good way to be on the go so fast and delicious , and some really were fun . ! They are smaller like .

#### SeqGAN with SELF

- This is my favorite . I am not a fan of flavored coffee , but the decaf was still a thick , bitter aftertaste you can buy
- I enjoyed this one drink when I paid 3 times hard . I like the flavor and it is very hard to find .

#### SeqGAN with LSTM

- I don't know if it's one of all time I have ordered anything I say I had some one dog that has this one food for it .
- These chips taste average . There have a very sensitive to anything , and I can't bring them pretty gross .

#### SeqGAN with IndRNN

- My son really enjoys a huge flavor of the instant Hot Cocoa mix , we have lots of all the other bags of coffee in this drink , but clearly solution price for him .
- I have been very disappointed with this product five stars , this can be of my first son . It's a good tasting tasting product.It , but I don't eat it because I like it .

## 5.4 Discussion

The implementation of GANs with different discriminators have shown, from the results, some improvements according to the metrics used. We have to consider that for the word-overlapping metrics, higher n-grams give a more significant measure because they compare the quality of longer sequences of words, which is a challenge for the model since it has to learn the dependencies between them.

By analyzing the BLEU and POS-BLEU score in the short text generation task, *SeqGAN with IndRNN* has obtained the best scores for every n-gram. While in the medium text generation task, if we consider the higher n-grams, *SeqGAN with SELF* has better BLEU scores over the others, but *SeqGAN with CNN* has better POS-BLEU scores. In the long sentences generation the results have shown better scores for the model *SeqGAN with LSTM*. The results show that in most of the cases, the adversarial training improve these results over the traditional RNN using MLE training, but we have to keep in mind that the baseline has been used without applying any relevant optimization technique.

We also notice that GAN models suffer from mode collapse according to the SELF-BLEU metric, an assumption is that after the models have found a way to fool the discriminator they tend to generate those samples with little modification.

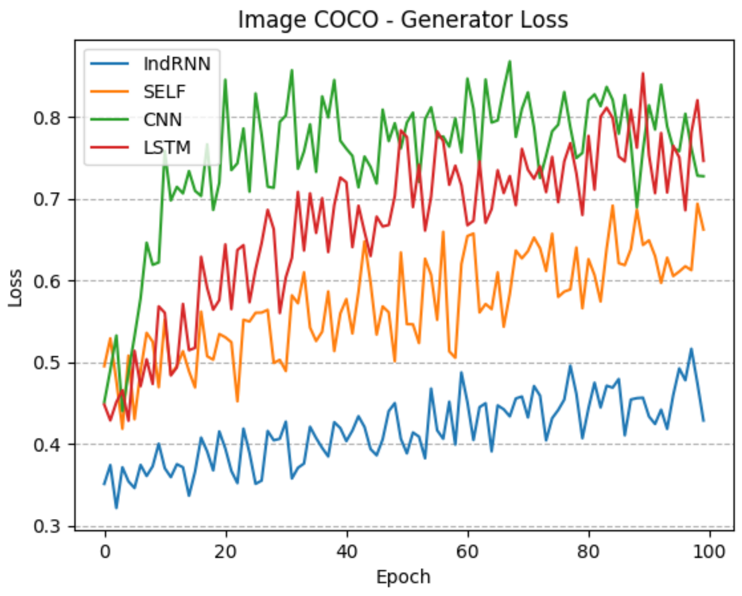
The readability metrics give us a hint of how complex is the generated text, in this case the generated samples from the GANs seem to have reduced the structure complexity compared to the test data. These metrics though may be limited due to the fact that their application imply that the sentences are semantically correct.

The loss charts show that, as the discriminators get more accurate the generator loss increase, the reason of this is that the generator is forced to explore other possibilities because the current generations are being classified as fake. In all the experiments, the CNN discriminator tends to become too accurate during the early stages of the training causing the generator loss to spike. From this situation, it seems that the generator is unable to improve itself in order to fool the discriminator, so both losses remain in the same range for the rest of the training. The discriminators RNN-based have a slower convergence, as a result the generator loss increases at a slower rate. In the end, by looking at the charts, in all the implementations, the generator is unable to fool the discriminator, as once the latter gets accurate the generator does not manage to improve itself in order to lower the discriminator accuracy, so the Nash Equilibrium is not reached.

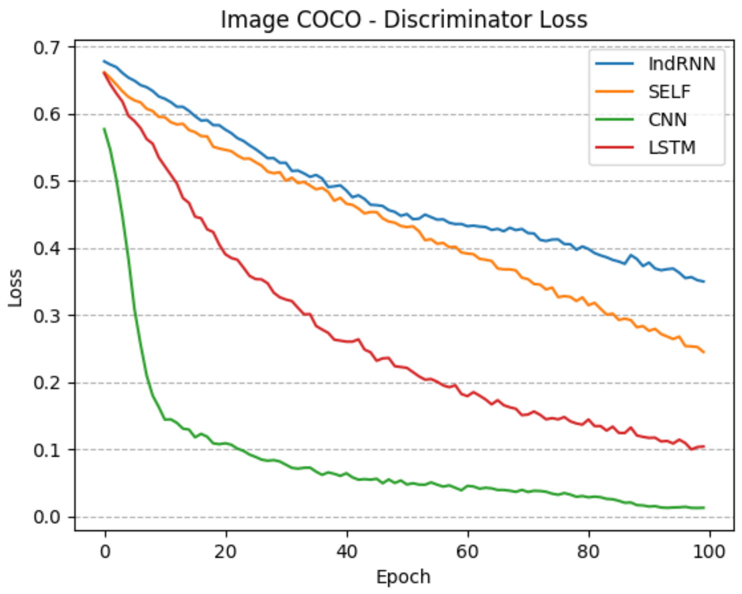
Despite some results obtained may appear positive, we have to remember that these metrics cannot fully evaluate all the aspects of the generated text. For instance, below the two sentences give similar results even though the first is taken from the training data while the second is generated and by looking at it, it does not provide a meaningful message:

- He told The Sun : ' I have waited long enough for this - it ' ll be a great start to the new year . [BLEU-4: 0.5539]
- It ' s been a very safe , but appear will a New Year ' s resolution at a time on the paper . [BLEU-4: 0.5527]

Not having exhaustive metrics is still a problem for the generative models based on GAN for the task of text generation. There is not a common clear evaluation system like the traditional models based on RNN language models, which use perplexity [66]

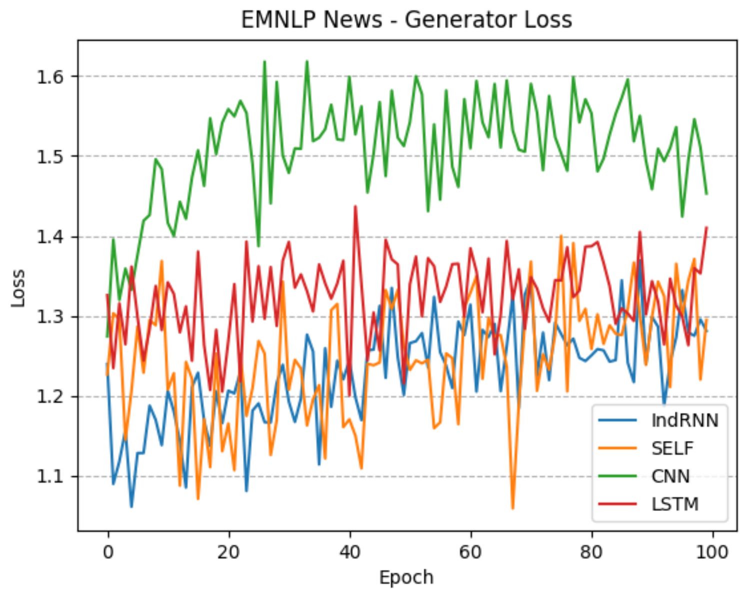


(a) Image COCO - Generator Loss during adversarial training

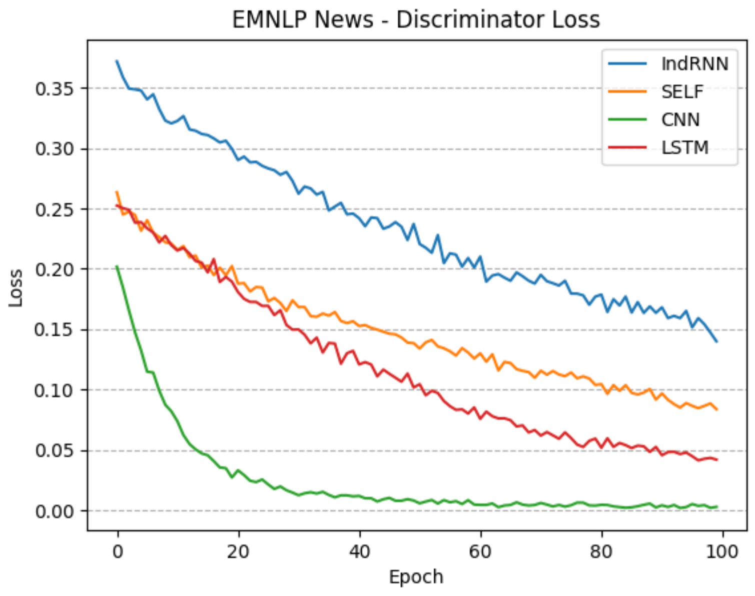


(b) Image COCO - Discriminator Loss during adversarial training

Figure 5.1. Image COCO - Generator and Discriminator loss

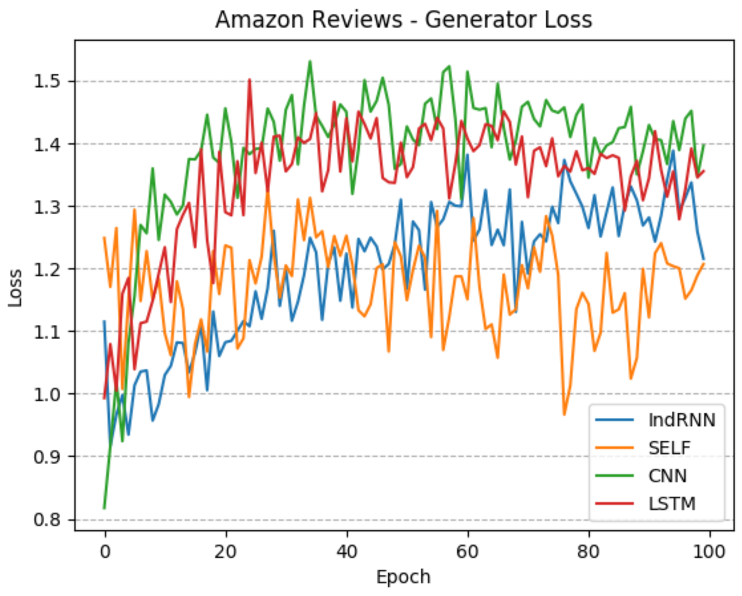


(a) EMNLP News - Generator Loss during adversarial training

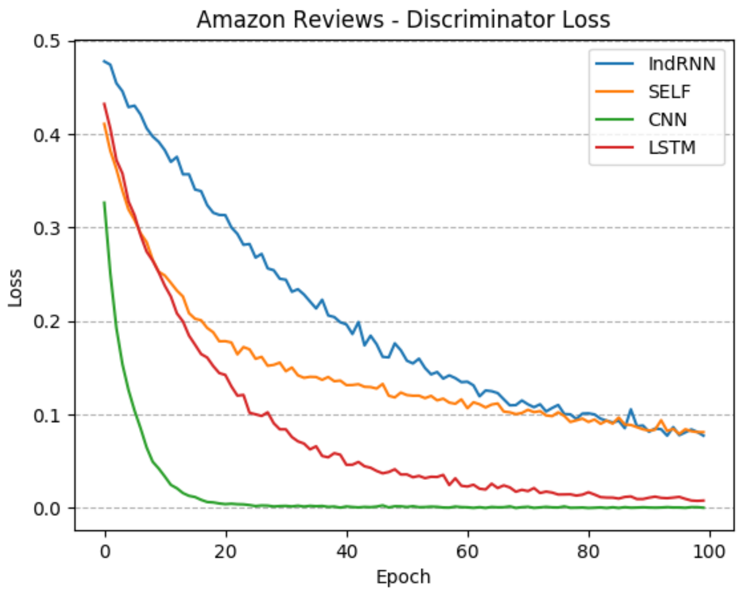


(b) EMNLP News - Discriminator Loss during adversarial training

Figure 5.2. EMNLP News - Generator and Discriminator loss



(a) Amazon Reviews - Generator Loss during adversarial training



(b) Amazon Reviews - Discriminator Loss during adversarial training

Figure 5.3. Amazon Reviews - Generator and Discriminator loss

# Chapter 6

# Conclusions

In this work, we show how generative adversarial networks can be used for the task of text generation. The main goals were to test the capabilities of these models to generate realistic text and evaluate the generated text with the available metrics. The architecture used is based on SeqGAN, which overcome the differentiability problem of discrete data by using REINFORCE algorithm. We experiment SeqGAN with four different implementations of the discriminator, three of which based on recurrent neural networks and attention mechanism to test whether they would give a better feedback signal to the generator compared to the CNN-based discriminator. We have seen that by using the same training setting, the results changed depending on the which discriminator have been used. In some cases the discriminators based on RNN obtained a better score than the one based on CNN. But further investigation is necessary to study the optimal hyperparameters for each implementation, in order to achieve the best score for each one with its own setting. A point to take into account is that we have trained the models for 100 adversarial training epochs, it may be interesting to analyze the losses behaviour with more training epochs to test whether the generator and the discriminator are able to reach the Nash equilibrium.

From the results, in particular from the generated samples, we have seen that SeqGAN model still need to be improved in order to generate realistic text. The best performance can be seen in the short generation task, where the model managed to generate some meaningful sentences. But in the other experiments the model showed not to be able to preserve global coherence throughout the sentence. As discussed in the previous chapter the metrics provide a comparison system between similar models, but are not always reliable, the reason is that they are based on word overlapping methods, which do not consider all the semantic aspects of natural language. Therefore, human judgment it is still necessary to accurately asses the text quality.

## 6.1 Future works

As future work, since we have used different discriminators and we have seen that they behave differently, we would like to use multiple discriminators, for instance a CNN and an RNN, to provide different feedback to the generator. By having a discriminator specialized at evaluating an entire sequence, while the other one in evaluating intermediate states the model could provide more information on how to correctly update the generator.

We would like to try to improve the language model by implementing it at character level, this method has been quite successful and could make the model less complex since the vocabulary is limited to the single symbols. Despite this advantage, we have to keep in mind that the model will also have learn how to generate the words other than the sentences. Another option would be to use a language model that is pre-trained on a large corpus and then apply a transfer learning technique in order to adapt the model to our datasets, after that we test whether the adversarial training would improve the model.

Recent researches, at the time of writing this thesis, introduced new metrics with which have shown that traditional RNN language models, if correctly trained with the appropriate optimization techniques, perform better than the GAN-based models [53, 59, 9]. Therefore, we would like to carry out more experiments to verify whether the adversarial training improve the results of an optimized RNN language model.

# Bibliography

[1] Arisoy, E., Sainath, T. N., Kingsbury, B., and Ramabhadran, B. Deep neural network language models. *NAACL-HLT* (2012), 20–28.

[2] Arjovsky, M., and Bottou, L. Towards principled methods for training generative adversarial networks. *ICLR 2017* (2017).

[3] Bahdanau, D., Cho, K., and Bengio, Y. Neural machine translation by jointly learning to align and translate. *arXiv:1409.0473* (2014).

[4] Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. Scheduled sampling for sequence prediction with recurrent neural networks. *arXiv:1506.03099* (2015).

[5] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. A neural probabilistic language model. *Journal of Machine Learning Research 3* (0 2003), 1137–1155.

[6] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks 5* (3 1994), 157 – 166.

[7] Bojanowski, P., Grave, E., Arm, J., and Mikolov, T. Enriching word vectors with subword information. *arXiv:1607.04606* (2016).

[8] Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. A survey of monte carlo tree search methods. *IEEE* (1999), 1–43.

[9] Caccia, M., Caccia, L., Fedus, W., Larochelle, H., Pineau, J., and Charlin, L. Language gans falling short. *arXiv:1811.02549* (2018).

[10] Charniak, E. Statistical language learning. *MIT press* (1996).

[11] Che, T., Li, Y., Zhang, R., Hjelm, R. D., Li, W., Song, Y., and Bengio, Y. Maximum-likelihood augmented discrete generative adversarial networks. *arXiv:1702.07983* (2017).

[12] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Schwenk, F. B., and Bengio, Y. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv:1406.1078* (2014).

[13] Danihelka, I., Lakshminarayanan, B., Uria, B., Wierstra, D., and Dayan, P. Comparison of maximum likelihood and gan-based training of real nvps. *arXiv:1705.05263* (2017).

[14] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. Language modeling with gated convolutional networks. *arXiv:1612.08083* (2016).

[15] David E. Rumelhart, G. E. H., and Williams, R. J. Learning representa- tions by back-propagating errors. *Nature* (9 1986), 533–536.

[16] Elman, J. Finding structure in time. *Cognitive Science 14* (2 1990), 179–211.

[17] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. *arXiv:1705.03122* (2017).

[18] Gers, F. A., Schmidhuber, J., and Cummins, F. Learning to forget: Continual prediction with lstm. *Neural computation 12* (10 2000), 2451–2471.

[19] Goodfellow, I. Generative adversarial networks for text. `https://www.reddit.com/r/MachineLearning/comments/40ldq6/generative_adversarial_networks_for_text/`, 2016. [Online; accessed 10-November-2018].

[20] Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial networks. *NIPS* (2014), 2672–2680.

[21] Guo, J., Lu, S., Cai, H., Zhang, W., Yu, Y., and Wang, J. Long text generation via adversarial training with leaked information. *arXiv preprint arXiv:1709.08624* (2017).

[22] Hochreiter, S. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness , Knowledge-Based Systems 6* (2 1998), 107–116.

[23] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9* (8 1997), 1735–1780.

[24] Huszár, F. How (not) to train your generative model: Scheduled sampling, likelihood, adversary? *arXiv:1511.05101* (2015).

[25] Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv:1611.01144* (2016).

[26] Kalchbrenner, N., Espeholt, L., Simonyan, K., van den Oord, A., Graves, A., and Kavukcuoglu, K. Neural machine translation in linear time. *arXiv:1610.10099* (2016).

[27] Karafiàt, M., Burget, L., Černocký, J., and Khudanpur, S. Recurrent neural network based language model. *INTERSPEECH* (2010).

[28] Kim, Y. Convolutional neural networks for sentence classification. *arXiv:1408.5882* (2014).

[29] Kim, Y., Jernite, Y., Sontag, D., and Rush, A. M. Character-aware neural language models. *AAAI 2741-2749* (2016).

[30] Kingma, D. P., and Ba, J. Adam: A method for stochastic optimization. *arXiv:1412.6980* (2014).

[31] Kneser, R., and Ney, H. Improved backingoff for m-gram language modeling. *Acoustics, Speech and Signal Processing* (1995).

[32] Kusner, M. J., and Hernández-Lobato, J. M. Gans for sequences of discrete elements with the gumbel-softmax distribution. *arXiv:1611.04051* (2016).

[33] Lamb, A., Goyal, A., Zhang, Y., Zhang, S., Courville, A., and Bengio, Y. Professor forcing: A new algorithm for training recurrent networks. *arXiv:1610.09038* (2016).

[34] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86* (1998), 2278–2323.

[35] Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. Adversarial learning for neural dialogue generation. *arXiv:1701.06547* (2017).

[36] Li, S., Li, W., Zhu, C. C., and Gao, Y. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. *arXiv:1803.04831* (2018).

[37] Lin, K., Li, D., He, X., Zhang, Z., and Sun, M.-T. Adversarial ranking for language generation. *arXiv:1705.11001* (2017).

[38] Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. L., and Dollár, P. Microsoft coco: Common objects in context. *arXiv:1405.0312* (2014).

[39] Lin, Z., Feng, M., dos Santos, C. N., Yu, M., Xiang, B., Zhou, B., and Bengio, Y. A structured self-attentive sentence embedding. *Conference paper in 5th International Conference on Learning Representations* (2017).

[40] Manning, C., and Schütze, H. Foundations of statistical natural language processing. *MIT press 999* (1999).

[41] McCulloch, W., and Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics 5* (4 1943), 115–133.

[42] Mikolov, T., Chen, K., Corrado, G., and Dean, J. Efficient estimation of word representations in vector space. *arXiv:1301.3781* (2013).

[43] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S. Recurrent neural network based language model. *INTERSPEECH 2* (2010), 3.

[44] Mikolov, T., and Zweig, G. Context dependent recurrent neural network language model. *SLT* (2012), 234–239.

[45] Mikolov, T., and Zweig, G. Context dependent recurrent neural network language model. *SLT* (2012), 234 – 239.

[46] Morin, F., and Bengio, Y. Hierarchical probabilistic neural network language model. *AIS-TATS'05* (2005), 246–252.

[47] Nash, J. Equilibrium points in n-person games. *Proceedings of the national academy of sciences 36* (1 1950), 48–49.

[48] PAPINENI, K., ROUKOS, S., WARD, T., AND ZHU, W.-J. Bleu: a method for automatic evaluation of machine translation. *Computational Linguistics* (July 2002), 311–318.

[49] PENNINGTON, J., SOCHER, R., AND MANNING, C. D. Glove: Global vectors for word representation. *Empirical Methods in Natural Language Processing* (2014), 1532–1543.

[50] POPOVIC, M., AND NEY, H. Syntax-oriented evaluation measures for machine translation output. *Proceedings of the Fourth Workshop on Statistical Machine Translation* (Mar 2009), 29–32.

[51] RUDOLF, F. A new readability yardstick. *Journal of Applied Psychology 35* (5 1948), 333–337.

[52] SCHUSTER, M., AND PALIWAL, K. K. Bidirectional recurrent neural networks. *IEEE TRANSACTIONS ON SIGNAL PROCESSING 45* (11 1997).

[53] SEMENIUTA, S., SEVERYN, A., AND GELLY, S. On accurate evaluation of gans for language generation. *arXiv:1806.04936* (2018).

[54] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958.

[55] SRIVASTAVA, R. K., GREFF, K., AND SCHMIDHUBER, J. Highway networks. *arXiv:1505.00387* (2015).

[56] SUTSKEVER, I., MARTENS, J., AND HINTON, G. Generating text with recurrent neural networks. *Proceedings of the 28th International Conference on Machine Learning* (2011).

[57] SUTSKEVER, I., VINYALS, O., AND LE, Q. V. Sequence to sequence learning with neural networks. *Neural Information Processing Systems* (2014), 3104–3112.

[58] SUTTON, R. S., MCALLESTER, D., SINGH, S., AND MANSOUR, Y. Policy gradient methods for reinforcement learning with function approximation. *NIPS* (1999), 1057–1063.

[59] TEVET, G., HABIB, G., SHWARTZ, V., AND BERANT, J. Evaluating text gans as language models. *arXiv:1810.12686* (2018).

[60] VEZHNEVETS, A. S., OSINDERO, S., SCHAUL, T., HEESS, N., JADERBERG, M., SILVER, D., AND KAVUKCUOGLU, K. Feudal networks for hierarchical reinforcement learning. *arXiv:1703.01161* (2017).

[61] WEIZENBAUM, J. Eliza–a computer program for the study of natural language communication between man and machine. *Communications of the ACM 9 6* (1966), 36–45.

[62] WERBOS, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE 78* (10 1990), 1550 – 1560.

[63] WILLIAMS, R. J. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning* (1992), 229–256.

[64] Yin, W., Kann, K., Yu, M., and Schütze, H. Comparative study of cnn and rnn for natural language processing. *arXiv:1702.01923* (2017).

[65] Yu, L., Zhang, W., Wang, J., and Yu, Y. Seqgan: Sequence generative adversarial nets with policy gradient. *AAAI* (2017), 2852–2858.

[66] Zhang, D., and Wang, D. Perplexity a measure of the difficulty of speech recognition tasks. *The Journal of the Acoustical Society of America 62* (1977).

[67] Zhang, D., and Wang, D. Relation classification via recurrent neural network. *arXiv:1508.01006* (2015).

[68] Zhang, Y., and Wallace, B. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv:1510.03820* (2015).

[69] Zhou, P., Shi, W., Tian, J., Qi, Z., Li, B., Hao, H., and Xu, B. Attention-based bidirectional long short-term memory networks for relation classification. *ACL* (2016).

[70] Zhu, Y., Lu, S., Zheng, L., Guo, J., Zhang, W., Wang, J., and Yu, Y. Texygen: A benchmarking platform for text generation models. *arXiv:1802.01886* (2018).