POLITECNICO DI TORINO

Master degree course in Computer Engineering

Master Degree Thesis

Language Generation Methods for Conversational Interfaces

Supervisor
prof. Maurizio Morisio

Candidates
Marco Bilardi
Student ID: 243911

Internship Tutor
LINKS
dott. ing. phd. Giuseppe Rizzo

Academic Year 2018 - 2019

Contents

Abstract

Text generation is becoming more and more important in many applications both on consumer and company side, for example with voice assistants and chatbots. The former ones need to provide answers to the questions they are asked, that needs to be built with a correct syntax and match the topic of the question, while the latter is nowadays used commonly for basic assistance, for example during technical issues with some service provided from a company, which is able to save money by using a bot for problems that do not require human intervention. In this work, we aim to build a model that is able to generate text starting from a subset of words extracted from a full sentence called seeds, compressing them to obtain a local context (the topic of the sentence) and building a new sentence with the knowledge acquired during the training. To do this, we implemented a Generative Adversarial Network based on a Sequence-to-Sequence model made of an Encoder and a Decoder, which is evaluated through a REINFORCE algorithm that uses the output of a Discriminator, whose task is to decide if a sentence is real or generated. We obtained good results over a small dataset (Image COCO) with short sentences, with our model providing better results over the actual state-of-the-art models considering quantitative metrics, while also evaluating qualitative ones with human evaluation.

Chapter 1

Introduction

Human society is based on natural language, as it allows people to communicate and understand each other: this basic skill is learned by every person quite simply, but it is instead a great challenge for a machine to understand and/or reproduce such language, as it is built to deal with numbers rather than words.
Natural Language Processing (NLP) is the branch of machine learning in charge of finding a way to develop systems able to analyze, understand and reproduce such language in order to allow an interaction between humans and machines.

1.1 Story of Natural Language Processing

This task has seen an exponential growth in the last 10-15 years, but it is actually a process that started long before. Already in the 50s Alan Turing proposed a test [6] which he called The Imitation Game, and then was named Turing Test after him: this test tried to answer the question "can machines think?", aiming to prove if a machine was able to produce thoughts. This was in the form of a game with 3 actors, A B and C, respectively a man, woman and an interrogator, which are in separate rooms and have different objectives: C needs to pick who is the man and who is the woman, A has to try to deceive C while B wants to help instead. The game is made up of a series of questions asked from C to the other actors, and based on their answers C has to produce a result. Turing then asks: what happens in one of the players is replaced by a machine? C will decide wrongly as often as it was with the two human players? If yes, we can reasonably assume that the machine is able to think at the point to fool the interrogator.
Specifically for NLP, several approaches were used in past years:

- Rule-based algorithms: they define rules to handle languages that are of easy interpretation, but the number of rules to be crafted by hand is not scalable and it is hard to code the various ambiguities and special cases of a language.

- Pattern Matching: given an utterance from a human, search for the most probable answer among the templates it has, and fills into placeholders informations specific to the utterance given. This approach has obviously no knowledge of context and/or language itself, and produce standard and repetitive sentences over and over

- Neural Networks Techniques: they are able to create a model that is able to generalize and reproduce what has been observed from a huge collection of real data. The algorithms and mathematics behind these systems are actually very old, but up until now machine power was not sufficient to support such complex mathematical operations related to such a huge quantity of data to analyze.

In this work we will analyze in depth the Neural networks applied in text generation, and in particular a new approach: Generative Adversarial Networks.

1.2 Goals

We would like to have a system that is able to generate text for different reasons, both for companies (for example generate readable reports based on data) or people(chatbots).
Current state of the art has reached through recurrent neural network a very good level of approximation but with some limitations: we have good performances on the syntax structure but on the other side we have difficulties in maintaining coherence through the sentence.
Generative Adversarial Networks brought a new way of approaching generative tasks, in the form of two RNNs confronting each other and improving each other as a result. They proved themselves very good in the computer vision domain, especially in application aiming at generating pictures of people and/or places that do not exist. The goal of this work is finally to verify if GANs can perform in text generation as well as they do in visual tasks. One of the main reason for focusing on GANs is that ideally they are less prone to merely copy what has been observed in the training, as they rely on a feedback given by a discriminator rather than one given by the difference real-fake data. In this work we will verify how GANs perform in the following task:

given some seeds words, they will generate a complete utterance containing information obtained from a basic context given by those seeds: this will be achieved by using an encoder-decoder generator and an LSTM discriminator.

1.3 Chapters

This work is organized as follows:

- Chapter 2: A focus on the state of the art both for NLP in general and text generation specific, but also on the models used for such tasks (RNNs, LSTM, GRU).
- Chapter 3: We present the models and configuration for them used in the experiments, showing different solutions we found and explaining why we made specific choice.
- Chapter 4: The experimental setup is described, including datasets and metrics used for evalutation.
- Chapter 5: Presentation of the results obtained by the experiments and discussion of various advantages and disadvantages of the implementation.
- Chapter 6: Conclusion on the work and possible future works

Chapter 2

Background and related work

This chapter gives a general background about the main approaches used today for text generation from two main points of view: one focusing on the language modeling and word representation, considering the problem from a linguistic perspective while the second one is more practical and involves the mathematical models used to achieve results theorized in the first part.

2.1 Language models

In order to generate and/or manipulate natural language, it is necessary a model that is able to approximate it: this is a statistical language model, which uses a probability distribution over sequences of words to predict which word w_i would be most likely to follow w_{i-1}. Considering a sequence of words $w = w_1, w_2, ..., w_i, ..., w_N$ of length N, where w_i is a generic word at position i, the probability of word i is given by the product of the conditional probabilities of the previous words as follows:

$$P(w_i) = \prod_{i=1}^{N} P(w_i|w_1, w_2, ..., w_{i-1}) \tag{2.1}$$

A language modeled as such will consider only previous words, but (especially in generation tasks) without a context it is not possible to distinguish between phrases and words that are similar but used in different situations. For this reason context is introduced as a variable c conditioning the final prediction, defining a different kind of language model defined as conditional, as follows:

$$P(w_i|c) = \prod_{i=1}^{N} P(w_i|c, w_1, ..., w_{i-1}) \tag{2.2}$$

This theory implied many computations for even not so long sentences, so it was applied using the Markov assumption:

> *A stochastic process has the Markov property if the conditional probability distribution of future states of the process (conditional on both past and present states) depends only upon the present state, not on the sequence of events that preceded it. A process with this property is called a Markov process.* A. Markov

which leads to the application of the theory by means of n-gram, which are sub-parts of n words inside a larger sentence: uni-grams (n=1), bi-grams (n=2), tri-grams (n=3) were often used, but they are insufficient to model a complex language because sentences often have long distance dependencies: Considering a sentence of 15 words and applying a tri-gram model, word at position 10 could have a correlation with word at position 5, but when using a tri-gram model only last 3 words are considered, losing in this way information. It was needed a different way of representing words/sentences, so we reached vector space models.

2.2 Vector space Models

It is an algebraic model for representing words (or sentences or even entire documents) as vectors, in which each dimension corresponds to a separate element: if the element occurs, then its value is non-zero. Considering multiple documents, representing them in a vectorial space means that we can compute what is called cosine similarity between them, which allows to determine how much the documents are close to each other and then, similar in some aspects rather than others. Not only that, the model is simple as it is based on linear algebra, but with some downside, such as poor performance on long documents (because of very large dimensionality) and loss of the order of elements inside documents. Several different applications have been made during years, with some of the most relevant being one-hot encoding, bag of words and word embedding.

2.2.1 One-hot Encoding

A word using this type of encoding is represented by an N dimensional array (with N being the number of distinct words inside of the document) and with the only possible values being 0/1 (where 1 is set to the index assigned to the current word and 0 in every other place). This representation brings many problems, first of all an exponential growing in size of the vectors, alongside with the loss of context and correlation between words.

2.2.2 Bag of Words

The main difference with One-hot encoding is that BoW remove the limit of a binary representation, allowing a repeated word to have more importance over a more rarely occurring word. This brings a major shortcoming, since very frequently occurring word would start to overwhelm other less occuring ones, also with the risk of not bringing valuable information (articles or other very common words) : for this reason, tf-idf is often used. Term Frequency Inverse Document Frequency is meant to rescale the number of times a word occurs by averaging over the set of documents available, so that a word occurring often in a single document or rarely in many documents would be overall balanced.

2.2.3 Word Embeddings

Models described earlier are characterized by very little information which is sparse inside of less meaningful data (few ones among many zeroes): with word embeddings we try to fix this problem by making these array dense and aggregating more information with less values. Not only that, dimensionality can be fixed allowing a better generalization for different documents rather than being based on their length. Also, with this space of values being continuous, similar words will be relatively close in the space allowing to maintain correlations between words. This allows us to use algebraic relationships between words through matematical methods, for example : emb(king) - emb(man) + emb(woman) = emb(queen)

Considering a vocabulary of only 4 words {a,b,c,d}

Words inside of documents 1-2	a b d	a c
One-Hot Encoding	a = [1 0 0 0] b = [0 1 0 0] d = [0 0 0 1]	a = [1 0 0 0] c = [0 0 1 0]
Bag of Words	a = [1 0 0 0] b = [0 0.5 0 0] d = [0 0 0 0.5]	a = [1 0 0 0] c = [0 0 0.5 0]
Word Embeddings (dim=3)	a = [-0.7 1.2 1.7] b = [0.08 0.15 -1.09] d = [-1.9 0.45 -0.67]	a = [1.01 -9 0.44] c = [1.11 -0.22 0.34]

Figure 2.1: Considering two documents, examples of possible encodings for the considered sentences.

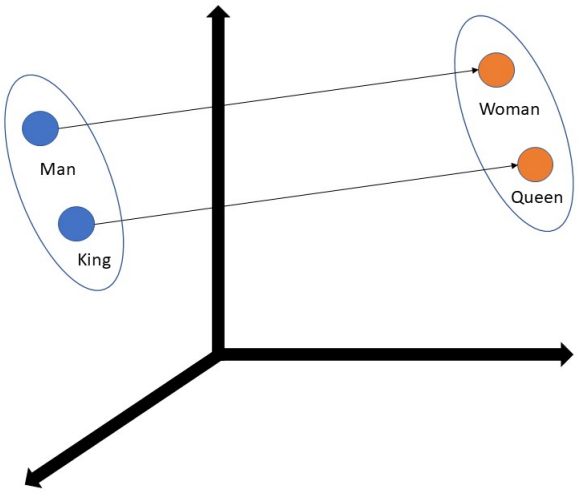

Figure 2.2: Example of algebraic relationships between words: the famous example King - Man + Woman = Queen.

2.2.4 Applications

Natural language processing groups together many different tasks that can be grouped in different categories, such as part-of-speech tagging related to syntax, communication tasks such as chatbots and speech recognition, or still sentiment analysis and text generation from the semantic point of view, the latter being the main focus of this work:

- Part-Of-Speech (POS) tagging: mainly used as a subtask for other bigger applications, final purpose of POS tagging is, given a sentence, to tag each word with its correct part of speech (noun, verb, adjective, etc...).
- Chatbots: built with the idea of simulating a conversation with a human partner, they have growing popularity for practical purposes such as customer service (a bot giving assistance and replying to customer questions) or vocal assistants (Alexa or Siri). They can have a various range of complexity, from simpler one (simply taking relevant words from a sentence and matching responses from a database) or more sophisticated ones.
- Speech recognition: determine words and sentences from an audio and transcribing them into text is a non so trivial tasks because of the diversity among the speakers, nonetheless can be important for example to people with disabilities.
- Sentiment analysis: target of this task is to infer subjective information from a set of documents, used often online to evaluate reviews about a product or to identify trendings among the public for marketing purposes.

2.3 Text Generation

Starting from a knowledge acquired from a set of documents, use those information to generate a text that is human readable. A model used for text generation can be modeled on different levels of complexity (character, n-gram, sentence, document) and is trained on a corpus to learn the likelihood of occurrences of a word i based on the previous words [0, i-1], generating a text very similar to the original one (allowing also to keep a style very close to the corpus one). This task is interesting as it is one of the most close to what can be defined machine creativity: could be used to generate scientific articles and entries for an encyclopedia, or even very human things such as

lyrics for songs or entire stories. The problem is modeled as a multiclass prediction problem, where at each time and given preceding history we need to select among all the entries in our dictionary what is the most probable word that will occur in that context and in that moment, based on what has been observed during the training. Vocabulary is made up of the words contained in the corpus and, with each of them having associated an index. The final aim of the task is to minimize the cross-entropy loss between the predicted output and the expected one as follows:

$$H(y^t, \hat{y}^t) = -\sum_{i}^{N} y_i^t log \hat{y}_i^t \tag{2.3}$$

where N is the number of entries in the vocabulary, y^t is the expected value for example t and $\hat{y}^t$ is the predicted value for the same example.

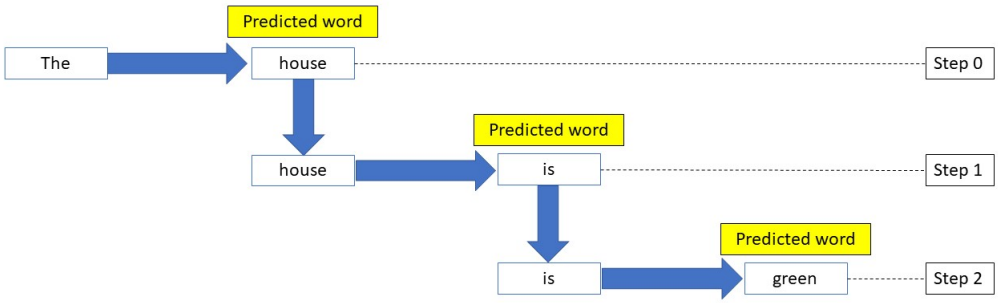

Figure 2.3: Generation step-by-step of the sentence “The house is green”.

2.4 Artificial Neural Networks

This kind of model is inspired by how the human brain works: they have the ability to learn through observation and experience, adapting themselves during the training process to generalize in the best way possible what is fed to them. They are called neural networks because they are made up of several cells called neurons, stacked up in both depth and width to make possible very complicate tasks impossible to achieve with linear networks. All of

the neurons inside a neural network are made up of a set of weights obtained through training and the activation function, which is the computation done on the incoming input to produce a result. They have an increasing popularity nowadays because of their flexibility and ability to recognize pattern and to generalize them in various fields such as computer vision and natural language processing, and because of the GPUs power allowing to stack many layers of neurons to create what are called Deep Neural networks.

2.4.1 Feed forward networks

Feed forward networks (FFNs) are the basic type of neural networks, featuring 3 main parts: an input layer, through where data is fed to the network, an hidden layer, where the neurons apply the activation function on the data received, and an output layer where results are found. The generic formula to describe this process can be

$$y = f(W * x + b) \tag{2.4}$$

where W is the weight matrix of the hidden layer learned through training, which will allow later to generalize any input data (also unseen ones), x is the input data and b is a bias term which is used to modify values that are slightly off.

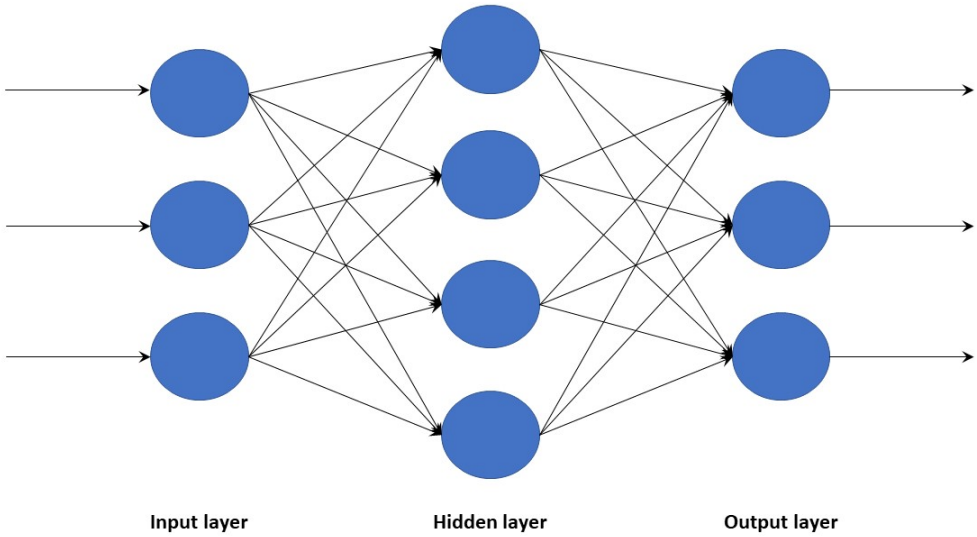

Figure 2.4: Example of structure and connections in a feed forward network.

2.4.2 Activation Function

When neurons receive a new input, they apply on it what is called an activation function: these are functions that, giving a certain input, maps it to a different value, often in range [-1, +1] or [0, 1] and have also some important properties such as being monotonic, a finite range of values where the function exists or is non-zero, and continuously differentiable.
Some examples could be:

- Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.5}$$

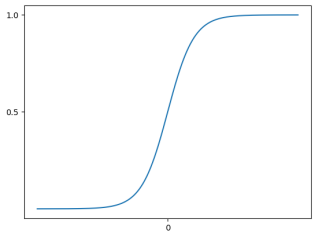

Figure 2.5: Sigmoid activation function.

- ReLu

$$f(x) = \begin{cases} 0 & \text{if x} < 0 \\ \text{x} & \text{if x} \geq 0 \end{cases} \tag{2.6}$$

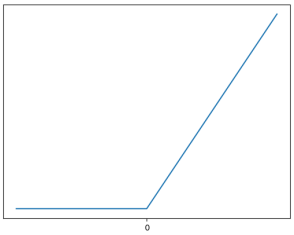

Figure 2.6: ReLu activation function.

2.4.3 Softmax

After obtaining the output from the neural network, since we're considering discrete data such as words, we need a way to discriminate among the available words what is the most viable for our current input: to do this it is used

a function called softmax, which creates a probability distribution over a set of classes (in our case words) having as total sum 1, and selecting the higher probability will give us the most likely class related to that input.
A standard softmax function considering K classes can be written as:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}} \tag{2.7}$$

2.4.4 Loss

The process called forward propagation, in which the input data is propagated through neurons, will generate an output more or less close to the expected value: how much these two results are close each other is called loss and is computed by means of a loss function. Loss helps the network to "learn" based on the evaluation it is given on a specific input, so it can re-adapt itself (by changing for example some weight of the neurons) to fit better what is the expected result.

2.4.5 Back propagation

Once the loss is computed, it needs to be inserted inside the network: this is done through back propagation, which computes the partial derivatives of the loss with respect to the weights, or gradients, which are then used to update neurons, in order to approximate always better the real model.

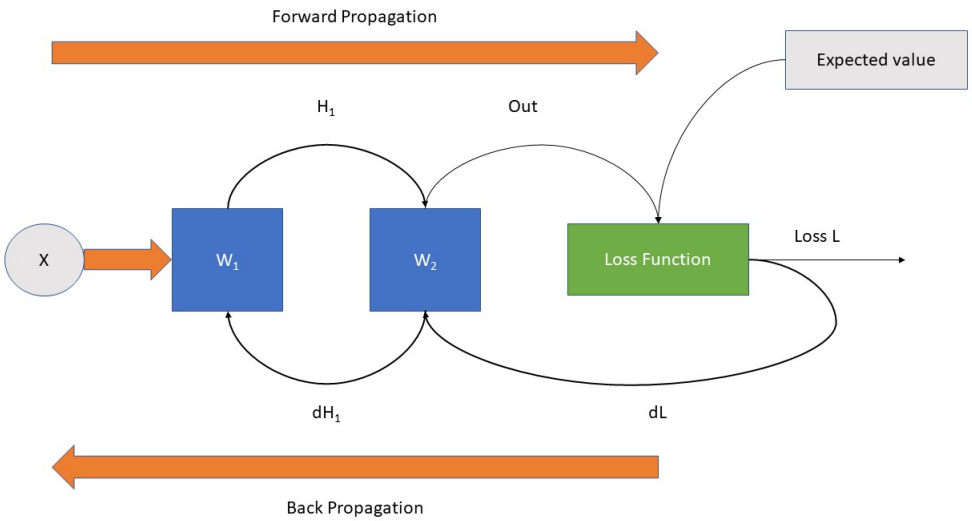

Figure 2.7: Example of error back propagation for two neurons W_1 and W_2.

2.5 Recurrent Neural Networks

Normal feed forward networks are limited with regards to persistence of the information acquired, since each input is computed separately from each other and no data is kept from them after the update of the neurons: recurrent neural networks try to address this problem, adding a loop inside that allows information to persist, having at each time step not only the current input but also the previous hidden state. This plays the role of something close to what we could call memory in a human brain and, because of this feature, they perform good in text generation having a way to maintain information over sequence of terms. However, there are multiple implementations of RNN with different features, the main ones considered here being Vanilla RNN, LSTM and GRU.

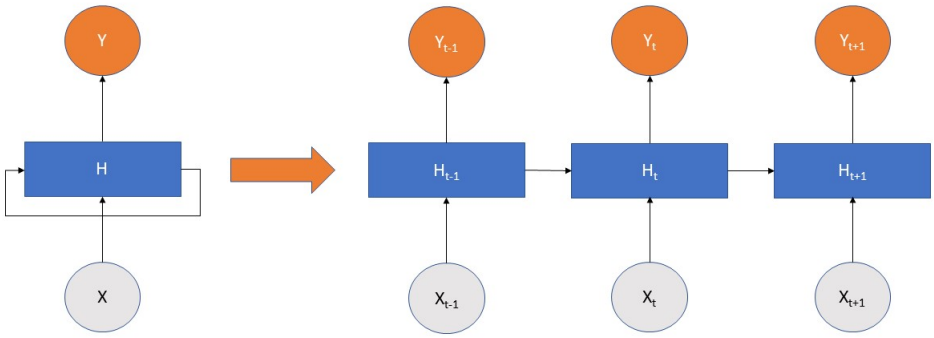

Figure 2.8: Unrolled recurrent neural network, showed as a sequence of steps.

2.5.1 Vanilla RNN

Most Basic implementations for rnn, it has three learnable matrix of weights U,W,V used in order for inputs, hidden states and outputs at each time step t:

$$H(t) = \sigma(U * X_t + W * H_{t-1}) \tag{2.8}$$

$$Y(t) = Softmax(V * H(t)) \tag{2.9}$$

This implementation has poor practical usage because of common problems related to the gradient (vanishing and exploding) that prevent a useful backpropagation of error: specifically, vanishing gradient presents values for gradients which decrease too much and, making gradients near zero means that the network stops getting better at a certain point. On the opposite, exploding gradient occurs when, for some reason, gradients become very large: this causes the networks to diverge and to make wrong decision on the parameters updates. Both of this problems were solved or mitigated with LSTM.

2.5.2 LSTM

LSTM meaning Long-Short Term Memory [1] has among its feature the ability to learn long-term dependencies better than a vanilla RNN by using a gating mechanism, which allows at each time step to evaluate and decide what informations should be forgotten or added to the memory of the network. Because of this, its structure includes the cell state C other than the classic hidden layer, used for storing past informations. LSTM operates as follows:

- Decide what information is going to be discarded from the cell state: to do this use a sigmoid layer called "forget gate layer" that, based on h_{t-1} and x_t, outputs a number in range [0,1] indicating whether or not keep part or any of the considered information. [2.10]
- Next decision is: what new information will be stored? First, a sigmoid layer called "input gate layer" decides the values to update [2.11] and then a tanh layer creates a new vector of candidates $\hat{C}_t$ that could be added to the state [2.12]
- Create now the new state: multiply the old state by f_t (forget part), then add $i_t * \hat{C}_t$ (input part) 2.13
- Last part is decide the output: this is done by applying a sigmoid on the cell state, filtering what needs to be kept or not, then multiplying this by a tanh layer applied on the cell state, obtaining the final output h_t [2.14]

$$f_t = \sigma(W_f * [h_{t-1}, x_t] + b_f) \tag{2.10}$$

$$i_t = \sigma(W_i * [h_{t-1}, x_t] + b_i) \tag{2.11}$$

$$\hat{C}_t = tanh(W_c * [h_{t-1}, x_t] + b_C) \tag{2.12}$$

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \tag{2.13}$$

$$o_t = \sigma(W_o * [h_{t-1}, x_t] + b_o) h_t = o_t * tanh(C_t) \tag{2.14}$$

2.5.3 GRU

GRU (gated recurrent unit) [2] is another way of improving vanilla RNN, it is very similar to LSTM as it uses a gated mechanism as well, but the memory here is managed entirely inside of the hidden state, with no existing cell state: this allows to have reduced number of parameters. Also, it has the input and forget gate combined together inside what is called the "update gate", it has been having increasing popularity.

$$z_t = \sigma(W_z * [h_{t-1}, x_t]) \tag{2.15}$$

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

$$\hat{h_t} = tanh(W * [r_t * h_{t-1}, x_t]) \tag{2.17}$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \hat{h_t} \tag{2.18}$$

2.5.4 Sequence to Sequence model

Using the models analyzed up until now, Google developed Sequence to Sequence model [7] with the aim to map an input to an output with possibly different lengths: an example could be machine translation, where very often translating from a language to another would result in a different number of words, or speech recognition [8]. This model consists of encoder and decoder, both being stacks of recurrent units (for example LSTM or GRU), the first has to compress information acquired from input and the decoder has to expand them starting from the encoder output:

- The encoder takes each element of the sequence x_i in input, collects informations and then produces an hidden state at each timestep.

- The last hidden states of the encoder will contain as much information as possible acquired from the input sequence, and this will be used as starting state for the decoder, allowing it to make more accurate predictions.
- Each unit of the decoder takes in input both the previous hidden state and word generated, producing itself the next hidden state and word predicted using informations acquired.

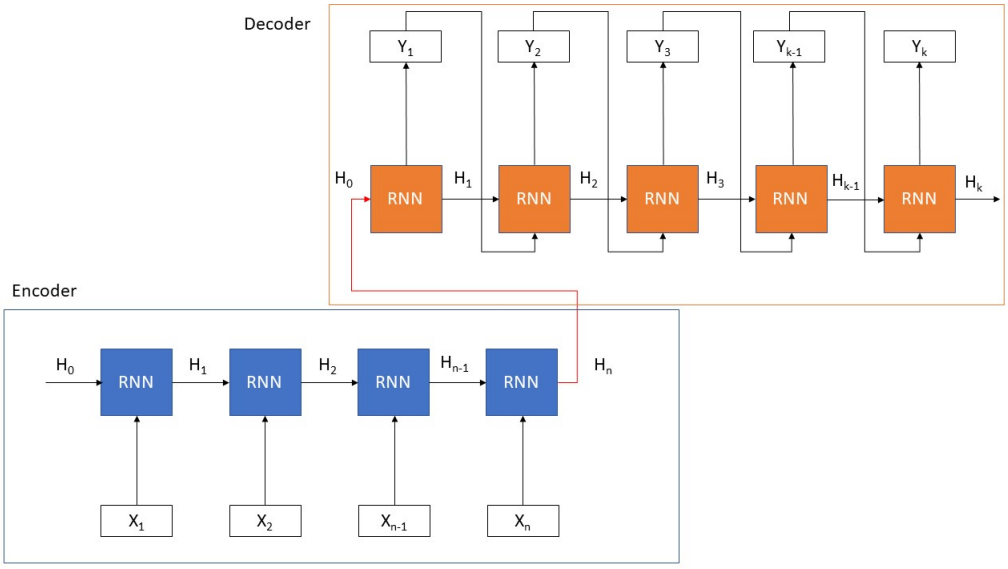

Figure 2.9: Example of a Sequence to Sequence model made of two parts, an Encoder and a Decoder.

2.6 Attention

This sequence model can be further improved using an attention mechanism as showed in [9]. The idea behind attention revolves around how human brain understands text: as we read text, we focus on specific parts of a document rather than others, in particular to those containing more valuable informations to identify and understand context, subjects and actors involved. This translates to a "choice" of what to consider more and what less, in other words we assign a weight to each word indicating how much that word is important inside of the sentence. It is particularly important considering long sentences, in which correlations between words that are distant each other could be lost, but with this mechanism this information is preserved.

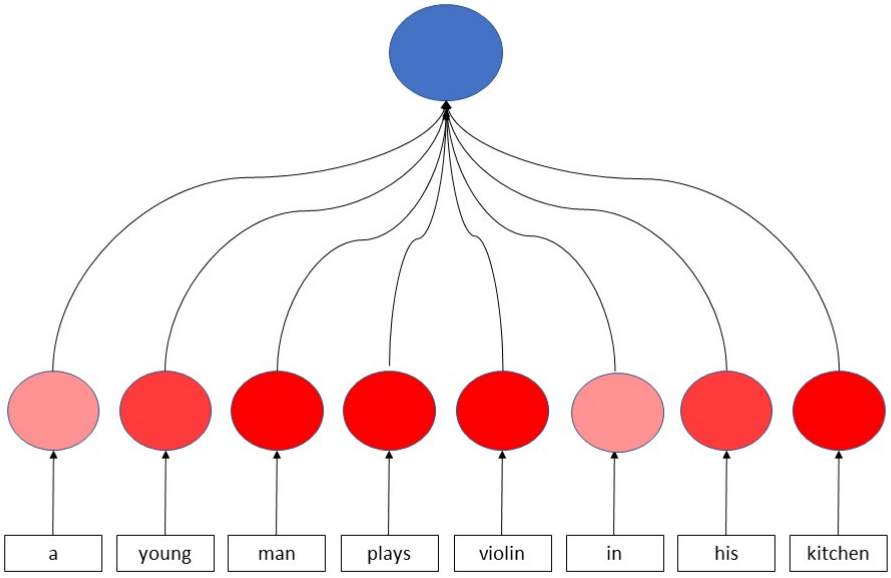

Figure 2.10: Intuition of how attention works based on color intensity: for each word, the color is deeper based on how much that word is relevant in the context of the sentence.

2.7 Generative Adversarial Networks

2.7.1 Game Theory

Game theory is the study of mathematical models for strategic interactions between decision-makers, which can be called players. It has many applications in social and computer sciences in a wide range of configurations, most common beings:

- Cooperative and non-cooperative: a game is defined as cooperative if the players are able to form alliance between external parties, while it is defined non-cooperative if parties cannot form alliance or they are self-enforced (meaning that the agreement cannot be created and/or destroyed by external parties): one of the most famous example is Nash Equilibrium.
- Symmetric and Asymmetric: a game is defined as symmetric if a payoff for a specific strategy depends only on the strategy itself and not on

who is playing it, meaning that a player i playing strategy s will have a different reward than player j using the same strategy.

- Zero-sum and non-zero-sum: in zero-sum games all player total rewards sum up to 0 for each strategies and at every step of the game, meaning that a loss to a player reflects a gain to another player: a very common example is poker. In non-zero-sum games instead, loss and gains are not complementary and do not sum up to 0.

- Simultaneous and sequential: in simultaneous games both players do their moves at the same time being unaware of the action performed by the other, while sequential ones allow each player to have some degree of information about earlier actions.

- Perfect and imperfect information: a game consists of perfect information if all players know the previous moves done by other players, but most games are made of imperfect information where players know no or some of the moves made by other

A game is made up of a set of steps made by the players, which are performed based on what is called decision rules: these are used to create a mapping between observation and action, and then evaluated through loss functions (how good was the decision taken?). One of the main rules used in this field is the minimax rule, used for minimizing (maximizing) the possible loss (gain) for worst (best) case scenarios. After a series of finite steps, we would like to reach a conclusion for the game, based on the previous moves observed: this process is called a solution, a formal rule that allows to predict the result of a game by deciding the strategies adopted by the players. Most famous proposed solution is certainly the Nash Equilibrium: if each player has chosen a strategy, and no player has advantages by changing only his strategy while maintaining the other ones unchanged, then this set of choices (and the corresponding payoff) are said to be in Nash Equilibrium.

2.7.2 GANs

Generative Adversarial Networks [3] are a new way of approaching generative tasks inspired by game theory explained before: at the base of this approach, we have a simultaneous training of two models, a generator G which generates new data based on what it has been observed in a real data distribution, and a discriminator D which estimates the probability that a samples came from the real world or from the generator output (fake). This process is modeled as a non-cooperative zero-sum game, in which G tries to fool D by producing

samples more and more similar to the real ones, and D has to discriminate between real and fake with growing precision. This will eventually bring to the desired result being the Nash Equilibrium, in which G generates examples indistinguishable from the real ones, and D basically does a coin toss on each sample (assigning 0.5 to both real and fake probability). This translates to reach a constant value for the cross-entropy in which 2 classes are considered (true/false) as follows:

$$loss(x, class) = -log(\frac{\exp(x[class])}{\sum_j \exp(x[j])}) \tag{2.19}$$

Starting from the definition of cross-entropy, in the equation we define x as the probability that an element is part of that class. The loss for each class is given by the negative log of the exponential of the considered class divided by the sum over all possible classes: since we have only 2 classes and we want them to have 0.5 probability each of occurring the equation reduces to:

$$loss(x) = -log(\frac{\exp(0.5)}{\exp(0.5) + \exp(0.5)}) = 0.69314 \tag{2.20}$$

This is the ideal value that the discriminator should reach when in Nash Equilibrium

2.7.3 Issues

However, these models have some major shortcomings, due to the fact of being highly susceptible to hyper-parameters choices and training instability, they end up very commonly in a state of non-convergence: this happens mostly when an update of G and/or D causes a wrong update on the other party. Another problem is found in vanishing gradients, specifically when D became too good to quickly recognize fake one: having its loss go down really quick, the error propagated back to G will tend to 0 causing the generator to stop learning. Last but not least is mode collapse, a problem which brings G to generate a very restricted set of outputs, in fact mapping many different inputs to the same output: this happens when G finds some pattern that fools D and updates its parameters to generate only that restricted set of outputs.

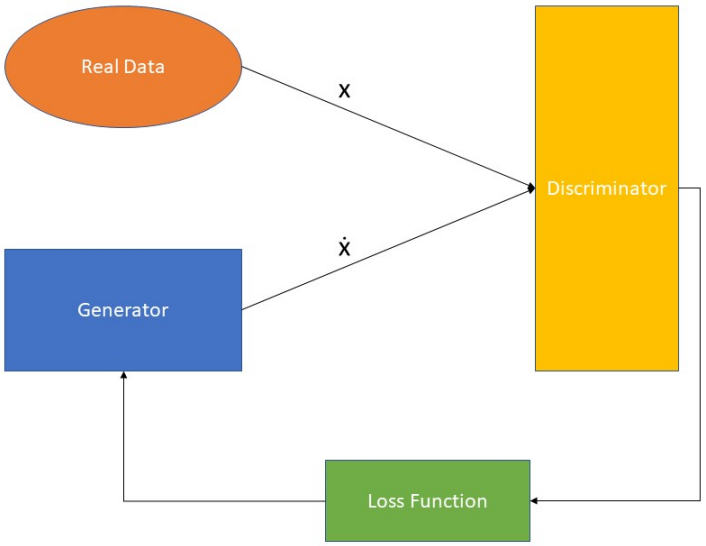

Figure 2.11: Basic GAN structure, with samples provided from the Generator and from Real Data, which are then evaluated by the Discriminator and the error computed is back-propagated to the Generator.

2.7.4 SeqGAN

In the specific task of text generation, the generator has to replicate content, style and semantic rules from the real dataset, while the discriminator tries to separate fake samples from real ones and provides feedback to the generator about the quality of the generated text. An important example of this task is SeqGAN [4], which is a sequence generation framework considering the problem of predicting the next word y_n taking into account the current state $s = (y_0, y_1, y_2...y_{n-1})$. The generator used is an LSTM which takes as input the sequence of embeddings $x_1, x_2, ..., x_n$ and produces a corresponding sequence of hidden states $h_1, h_2, ..., h_n$, then a softmax layer gives the probability distribution for the output. On the other side, the discriminator used is a CNN (convolutional neural network) [5] used to minimize the cross-entropy loss between ground-truth and prediction.

2.7.5 Reinforcement learning

Reinforcement learning is concerned with how agents take action inside of a certain environment in order to maximize some kind of cumulative reward

(at each timestep t we add a partial reward for the current action taken) based on some kind of policy: they are maps given the probability of taking action a gives state s. These policies are explored by different methods:

- Brute force: as the name says, for each possible policy, take the one with the maximum expected return. This is intuitively the best option but the number of policies can grow very large or even be infinite, also the return values can have high variance and then bias results.

- Value function: it attempts to find a policy that maximizes the return while maintaining a set of expected values obtained by some policies (for example the current one or the best one found until now). Here optimality is defined in a stricter sense than before, as a policy is here called optimal if it achieves the best return from any initial state.

- Monte Carlo Search: as the name says, it is based on Monte Carlo methods and relies on repeated random samplings to obtain results. To evaluate a policy using MC, given a policy π, a state s_0 and an action a_0, the goal is to compute the function $Q^{\pi}(s, a)$ for all state-action pairs starting from state s_0 and taking action a_0 until reaching an end to the decision tree and then averaging over different iterations of these process. This is a good approximation but it brings with it some problems such as slow convergence with high variance and possible time loss in evaluating a sub-optimal policy

2.7.6 Policy Gradient

From a practical point of view, while GANs perform really well with continuous data such as pictures, they have some limitations when related to text generation because, given the nature of languages, they are modeled as discrete sequences of tokens, making it difficult to back-propagate gradients to the generator: to deal with this problem, SeqGAN uses Policy gradient based on reinforcement learning, allowing the generator to take actions with high reward. Specifically, it uses Monte Carlo Search with a rollout policy: given a sample of n words, for each word w_i it generates w_{n-i} words, and then feeds these new sequences to the discriminator and, based on its output, assign a score to word w_i suggesting how much that word makes the sentence real: the final reward is averaged over all iterations of the algorithm.

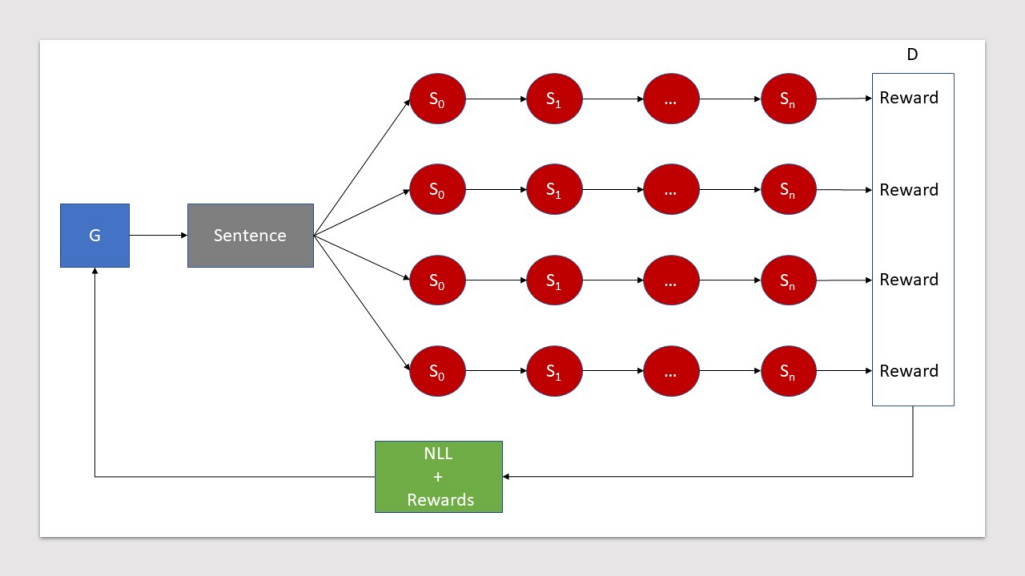

Figure 2.12: SeqGAN structure, where G is an LSTM and S_i are the steps for computing the reward for each sentence using Monte Carlo Search, reward which is back-propagated together with the NLL loss.

2.8 Important GAN applications

GAN saw different implementations for different tasks over the years, here we present some of the most important ones:

2.8.1 MaliGAN

Maximum-Likelihood Augmented Discrete Generative Adversarial Networks (MaliGAN) [17] addresses the problem of backpropagation in GANs by using the same architecture proposed in SeqGAN but modifiying the reward computation by rescaling it based on the batches of words with a specific length. This process creates an artificial constraint that do not allows the model to adapt efficiently to different document length, but allows for a better backpropagation of error to the generator.

2.8.2 RankGAN

Adversarial Ranking for Language Generation [18], or RankGAN, focus on the binary restriction that a discriminator has when evaluating outputs from GAN (either the result can be true or false, 0 or 1), explaining how this simplistic way of classification limits learning capabilities in a model that

needs to generate complex structures such as natural language processing. They address the problem by using a rank system over a collection of human-generated sentences and machine-generated ones, which analyzes sentences and assign a relative score to them, allowing the discriminator to do better estimations and helps giving back to the generator more useful information. One of the greatest problem in GANs applied to text is the backpropagation of the loss because of the discrete elements (tokens), and this paper makes a concrete step forward in solving this problem.

2.8.3 LeakGAN

GANs applied to natural language processing tasks have shown promising results in multiple tasks, but also they have a clear limitation over long sentences: the discriminator can provide feedback only when the entire text has been generated and so the generator is missing a support in the middle of the sentence. Long Text Generation via Adversarial Training with Leaked Information [19], shortened in LeakGAN, tries to solve this problem by introducing a leak in the discriminative model D, which allows for the generator G to obtain informations about features learned by D. This approach is very powerful especially in generating long sentences and/or text, which still now it is a non trivial task. This approach could be a great integration for our model, as we had to introduce a scale factor over long sentences to reduce the impact of bad tokens generated as the sentence grew longer, so adapting a similar feature could help us solving this problem.

2.8.4 TextGAN

Adversarial Feature Matching for Text Generation [20], or TextGAN, implements GAN model by using an LSTM as generator and a convolutional network as discriminator, using a custom metric based on Maximum Mean Discrepancy between the distributions of real and fake sentences, rathen than using GAN objective function: this approach allows to improve diversity and reduce mode-collapsing, which is a recurring problem in GAN applications.

2.8.5 MaskGAN

MaskGAN: Better Text Generation via Filling in the_____ uses a totally [21] different approach: while state-of-the-art models are based on maximum likelihood estimation and teacher forcing, also they are autoregressive in a way that they produce an output and use that as input in the next time step, making them more error prones, while MaskGAN implements a model

trained to fill the blanks in a sentence by taking all the known words and extracting a context from them. Not only that, it is not based on validation perplexity, which is no really indicative of the quality of text generated and so it misleads results. This approach is limited as it requires an almost complete sentence to produce an accurate prediction, transforming the generation task to a more simplistic completion task.

2.8.6 Language GANS Falling Short

Natural language processing tasks started by using maximum likelihood estimation based models to try learning how to compose sentences: this brought several problems due to teacher forcing during training, which resulted in what is called exposure bias during inference. To solve this problem we had GAN models take over with different solutions, but while they performed well in quality of sentences they also did poorly in diversity (mode collapse problem): Language GANs Falling Short [22] investigates that models based on MLE outperforms consistently models based on GANs by fine tuning the temperature parameter in the softmax function, while also being easier to train and computationally lighter. They prove themselves superior by maintaining a high quality with a good diversity, both aspects that often GANs fall short on.

Chapter 3

Approach

In this chapter we will explain the approach to text generation problem and the models used to implement such an idea. Starting from the intuition behind this work, we dive deeper into the models and choices done in matter of configuration and/or parameters used for training, finally discussing how we infer our model.

3.1 Task Description

The idea behind this work is very simple: starting from some knowledge, we would like to generate text able to express that knowledge in a human-readable text: the first challenge is how to encode in a meaningful way the acquired knowledge, and finally to produce text maintaining syntax and semantic structure learned from the real dataset. To achieve the first goal we decided to extract some specific information from each sentence that we called seeds, in the form of a triple made of Noun-Verb-Noun which should represent the backbone of the sentence(context, subjects, actions) that will be then expanded into a full sentence as follows:

Seed to text examples	
Seed	**Full sentence**
group driving motorcycle	A group of motorcycle riders driving past buildings.
dogs stick heads	Dogs stick their heads out of car windows.
man takes picture	A man takes a picture of an airplane taking off.
governor take decision	The governor would take an important decision for the United States.
Coliseum is amphitheatre	The Coliseum is an oval amphitheatre in the centre of the city of Rome, Italy.
movie is sequence	A movie is a medium composed of a sequence of images and sounds used to communicate ideas, stories and perceptions.

Table 3.1: Examples of text generation: on the left the seeds giving a context and to the right an example of possible output.

This process is implemented with a Sequence-to-Sequence model, where the encoder has the task of produce a context for the input sequence and then passing it down to the decoder, which will use it to produce an output sequence starting from the received context. Since we would like to see sentences which are very different from the ones observed, otherwise we would have a simple sequence prediction problem, we use GANs for this task which will allow us to use a discriminator feedback to judge the quality of the produced sentences.

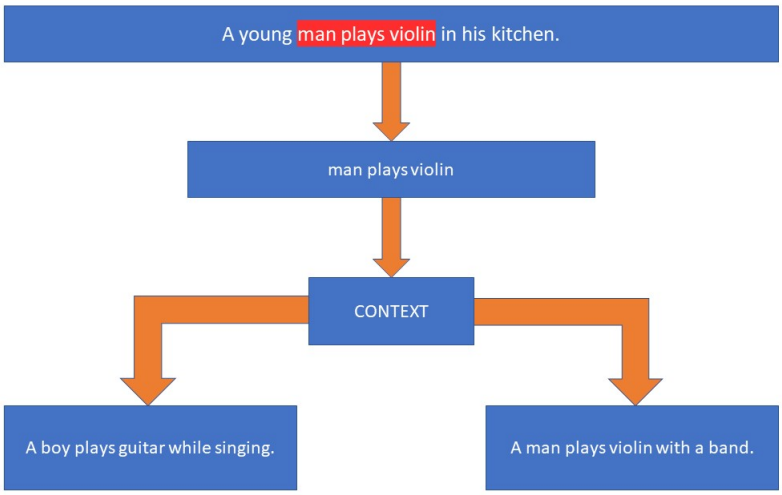

Figure 3.1: Steps for the generation: considering the sentence in figure, we extract three words (Noun-Verb-Noun), which will be used to obtain a context based on what has been learned through the training. The context is then used as the starting point for generating countless possible sentences, for example the left one (less close to the original seed but with a similar meaning) and the right one (closer to the original also in the words used).

3.2 Generator

As already specified, its task is to generate sentences starting from seeds words used to infer context and other informations useful to build the final sentence. The encoder, modeled with a GRU as well as the decoder, receives in input seed words and produces an hidden and an output layer: these are then used in an attention layer which allows to build a context allowing the decoder to predict the correct word avoiding ambiguities. What the decoder takes in input is in fact the concatenation of the current word embedding with the context information, producing then the most probable word w_N given context c and word w_{N-1}

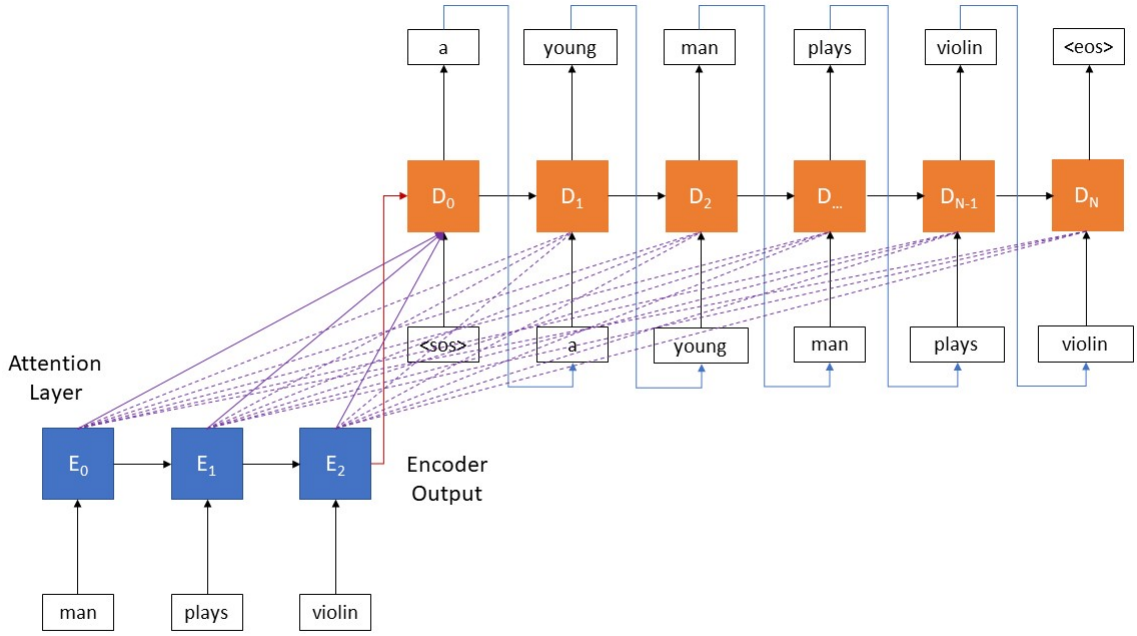

Figure 3.2: Steps for generating of the sentence "a young man plays violin in his kitchen": Encoder takes the seed as input, producing an output and intermediate hidden layers, which are then passed to the Decoder in the form of attention. This gives a context to the decoder, which starts to predict the next word based on the informations acquired plus the previous word (<sos> for the beginning) at each timestep.

3.3 Discriminator

The discriminator used is an Attention-based bidirectional LSTM introduced by [10], which is not affected by the vanishing gradient problem because of the bidirectional feature. It is used as a classifier for our generator in charge of discriminate between real data and generate one. It works as follows:

- Generate the embeddings starting from the input sequences.
- LSTM takes the input and produces the hidden states $H = h_1, h_2, ..., h_n$ where n is the sentence length.
- The attention layer assigns weights for each part of the sentence based on $\alpha = softmax(w^T * tanh(H))$, with w being a learned matrix weight.

- The context vector is created as $r = H * \alpha^T$.
- At this point the sentence representation is obtained as $h* = tanh(r)$.
- Finally a softmax layer is used to obtain probability distribution for output classes (in our case two).

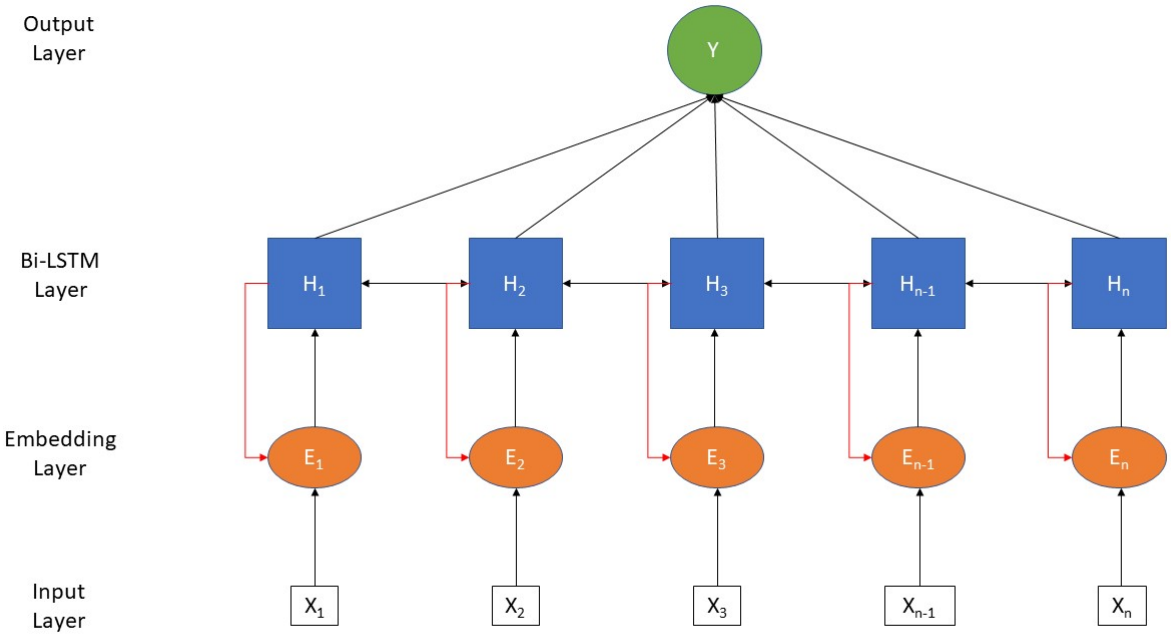

Figure 3.3: Bidirectional LSTM discriminator.

3.4 GAN Policy

Once defined generator G and discriminator D, we need to define how they will work together, where the main focus will be how D propagates error back to generator: this is done through a rollout policy as specified in [4], using the Negative-Log Likelihood (NLL) Criterion paired with the rewards given by the Monte Carlo Search. Starting from a sample X_g of length n generated by G, we generated $k = n - 1$ new samples X_r of same length starting from the first n-k terms given by X_g: in this way we evaluate how much each part of the sentence contributes to make the sample real. An example of length 5 is provided as follows, where x_i are words taken from sentence X_g given by G and r_i are words given from sentence X_r given by the Rollout policy:

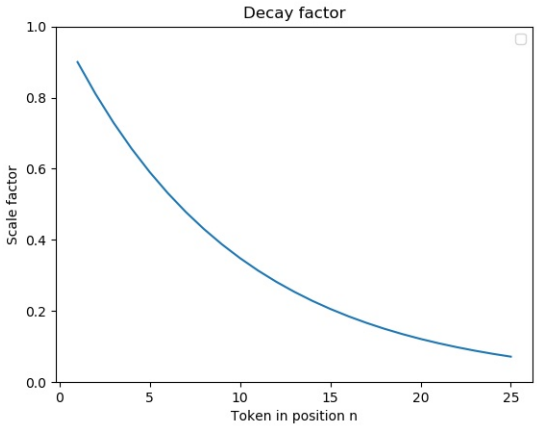

Figure 3.4: Progression of the decay factor as the sentence grows longer.

Iterations	
Time Step	Sample
1	$X_{r1} = [x_1, r_1, r_2, r_3, r_4]$
2	$X_{r2} = [x_1, x_2, r_1, r_2, r_3]$
3	$X_{r3} = [x_1, x_2, x_3, r_1, r_2]$
4	$X_{r4} = [x_1, x_2, x_3, x_4, r_1]$
5	$X_{r5} = X_g = [x_1, x_2, x_3, x_4, x_5]$

Table 3.2: Building of the sentences used for Monte Carlo search in rollout policy: considering a sentence of length k, at each timestep n, we take n tokens from the generated text, and k-n new tokens are generated. After k total tokens are obtained, we evaluate them through the discriminator and obtain the probability of the n tokens being part of a real sentence.

D evaluates these sentences and gives a score to each sentence in the form of probability that the sample is real. Because of the unstable nature of Monte Carlo search, we also apply a decay factor to the rewards to make the scores more numerically stable over the various iterations of the algorithm. Complete formula is described in Equation (3.1).

$$Loss(x) = \frac{1}{N} \sum_{j}^{N} NLL(x_j) * K^j * R_j \tag{3.1}$$

where x is the sample considered of length N, x_j the j-th word in the sample, K^j is a decay factor which decreases by 0.9 for each token contained in the sentence and R_j is the reward given to the j-th word.

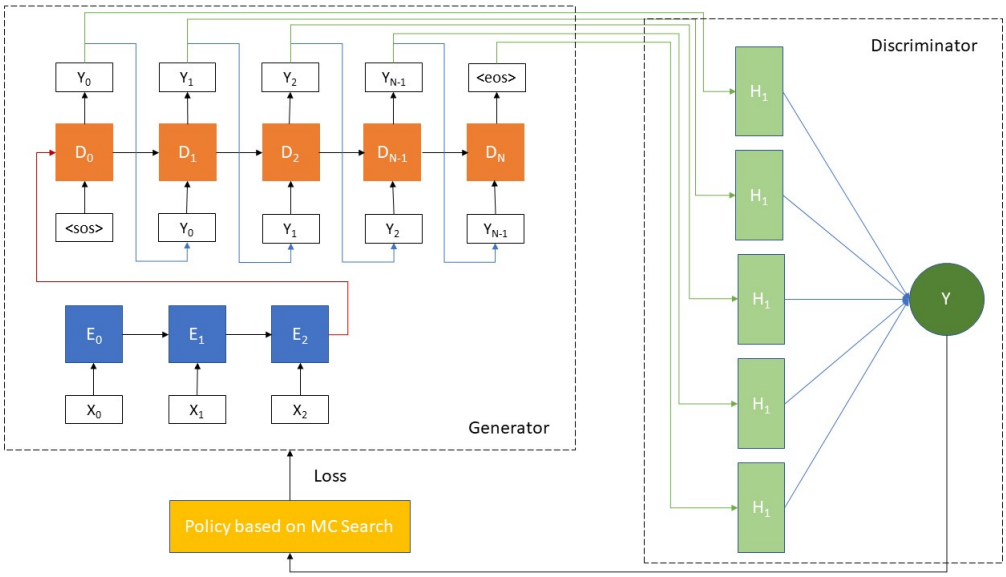

Figure 3.5: Complete view of the model, made up of Generator on the left and Discriminator on the right.

3.5 Training Process

With all the elements defined, we describe how a training scenario would be done with our model: first we pre-train both G and D because otherwise we would have a generator extremely weak and the loss for D would be too low since the start, not allowing to backpropagate a proper feedback, and at the same time D would be fooled to easily from the start. After this initial phase we begin the GAN training where we alternate a phase of training for G, the policy evaluation and lastly a training for D, where with this sequence of steps we would like to maximize the rewards in order to make the generated sentences indistinguishable from the real ones.

3.6 Inference

Once the training of the model is done, we can infer the model by submitting a seed of 3 words in the form of Noun-Verb-Noun, and generate an output sentence which describes an action defined by the verb involving the nouns. The seeds given as input do not necessarily define the exact words the sentence will contain, but they will define the context that will be used for the generation: for example considering the sentence already used as example “A man plays violin in his kitchen” the seeds “man,plays,violin” could bring the

Algorithm 1 Training Process

Initialize G_θ and D_ϕ with pre-trained GloVe.
Pre-train G_θ on real seeds X_r using Cross Entropy with the target sentences
Generate samples X_g with G_θ
Pre-train D_ϕ on real data X_r and fake data X_g using Cross Entropy
Define N as number of adversarial training epochs
for $i = 1, 2, ..., N$ **do**
    Feed seeds to the encoder E_θ
    for $k = 1, 2, ..., maxlen$ **do**
        Feed to the decoder D_θ the encoder output and token k
        Output token k+1
    end for
    Generate sequence X_g using G_θ
    Evaluate X_g using policy gradient as described in 3.4
    for $l = 1, 2, ..., rolloutnum$ **do**
        Generate new sentences X_p starting from X_g tokens
        Discriminate using D_θ and get loss for each token
    end for
    Compute loss l_g using equation (3.1)
    Update G_θ by backpropagating l_g
    Train D_ϕ on both X_g and X_r
end for

context in other general directions, for example involving men, instruments, games, and other things which could be more or less close to the context.

Algorithm 2 Inference

1: Load models G_θ and vocabulary V
2: Receive seed S in input
3: Obtain seed in form of word IDs S_i for the seed using V
4: Give S_i to the Encoder E obtaining its output E_o and starting hidden state for Decoder D $E_h = D_h$
5: Define starting word W_0
6: **for** $i = 1, 2, ..., L$ **do**
7: Feed D_h and E_o and word W_{i-1} to D
8: Predict next word W_i
9: Replace D_h with the newly computed hidden state
10: **end for**

Algorithm 2 gives an example of inference stage: this process is called "Free Running" as we don't deal anymore with real data but we use the collected knowledge to generate new sentences based on what has been observed.

Chapter 4

Experimental Setup

In this chapter we will present the tools and setup used for our experiments and evaluation, such as configuration parameters, datasets and metrics.

4.1 Dataset

For our experiments we used the Image COCO dataset [11], a dataset made up of annotated images used normally for object detection and captioning. Each sample is in the form: image caption describing the action/actors of the picture. Since our application only works on text, we stripped away the pictures. We use a subset of the total samples, specifically 9,000 sentences for training and 6670 for test, all of them tokenized using NLTK tokenizer:

1. Suppose to have the following sentence: "a young man sitting on the ground next to a skateboard."

2. Using Spacy we can extract a seed of three words which will create the context of the sentence (in the form of name-verb-name), in this case it would be: "man sitting ground"

3. At this point we tokenize the original sentence obtaining: "a young man sitting on the ground next to a skateboard ."

4. Put <eos> token to define the end of the sentence and <pad> to reach a pre-defined length (all the sentence need to have the same length for computation purposes): "a young man sitting on the ground next to a skateboard . <eos> <pad> <pad> <pad>"

5. The sentence is ready for being processed.

Image COCO dataset	Train set	Test set
Number of sentences	9,000	6,670
Number of tokens	95,679	69,276
Number of distinct words	4,625	3,455
Average sentence length	10.63	10.38
Max length sentence	35	32
Min length sentence	7	6
Max occurrences for word	14,919	10,893
Min occurrences for word	1	1
Words occurring only once	2,050	1,603
GloVe 42B coverage	99%	98.9%

Table 4.1: Relevant statistics for the dataset used.

4.2 Tools

We used PyTorch (version 1.1.0), an open source machine learning framework based on the Torch library and developed by Facebook's artificial intelligence research group. It implements in a transparent way tensors in a python environment, allowing most common tensor operations and deep neural networks models. We also used Natural Language ToolKit (NLTK) library for language-related tasks, such as tokenization and evaluation. Lastly we use Spacy to identify entities (names, verbs) used to generate seed words that give context for the final sentence. Lastly, we used GloVe pre-trained embeddings [12], created with a training on different Wikipedia articles, which will be adjusted during training adding information received from our datasets. We use the 42B version, containing 42 billion tokens and 1.9 million distinct words.

4.3 Configuration

In this section we will give details about the parameters used for training the model.

Parameter	Value	Explanation
General		
Epochs	50	We train the model for 50 epochs, since we observed from experiments that this time is more than enough to make the model converge.

Batch size	32	Indicates in how many subparts the dataset is split during the training, we chose a low value because we considered a relatively small dataset.
Embeddings size	300	Dimension of the vectorial space in which each word is placed based on the information acquired during training, we chose a high value to contain more information possible within each word.
Max sequence length	20	Max possible length for a sentence, value chosen based on the dataset considered, since we have few sentences longer than 20 words.
Dropout	0.3	Indicates how many cells are turned off randomly at each iterations during training, to make them less dependent on each other.
Learning rate	1e-4	Coefficient indicating how much we move our parameters at each iteration.
Policy	MC search	We used policy gradient from [4] but with a decay factor to give less penalty to last tokens over long sentences.
Optimizer	SGD	As observed in [13] methods based on gradient descent allows to generalize better a model and to obtain superior results over test set, while being slower to train.
Generator		
Pre- training	15 epochs	Number of iterations over the dataset for pre-training the generator, we chose a low value to be less biased during the adversarial part.
Model	GRU	We chose a bidirectional Gated Recurrent unit with 3 layers.

Hidden size	128	Number of cells being part of each layer inside of our network, we supposed we would not need more than this considering we also have a bidirectional network.
Optimizer	Adam	Adaptive algorithm which server the purpose of giving a good starting point for the generator, being also fast in execution.
Discriminator		
Pre-training	2 epochs	pre-training made for the discriminator as well as the generator, but we kept it even smaller here to avoid the discriminator getting too accurate since the beginning for training purposes.
Model	LSTM	We used a 1-Layer bidirectional LSTM with attention.
Label error	0.15	We perform label swapping in 15% of the dataset to slow down the discriminator and allowing the generator to learn from its feedback.
Optimizer	SGD	Stochastic Gradient Descent provides a slower convergence over GAN and allows to maintain a weaker discriminator which can backpropagate a better feedback.

Table 4.2: Main parameters used for the training of the model, from the left: Name of the parameter, value used, and brief explanation of meaning and reason of choice.

4.4 Metrics

A problem yet to be fully solved is how to evaluate generative models: how to evaluate complex aspects of language such as sentence coherence, grammar, syntax in a way similar to a human? The current standards are based on word-overlapping methods, word embedding and readability analysis. The latter is defined as the effort needed to read a written text, depending on

many factors such as the vocabulary and syntax. Unfortunately known metrics still fall short to what a real person evaluation could give, but are still a good baseline to start with: in particular we used BLEU [14], POS-BLEU [15] and SELF-BLEU [16].

4.4.1 BLEU

Bilingual evaluation understudy [14] is an algorithm designed for quality evaluation of a text translated from a language to another: it needs a human-written corpus to be used as reference and outputs a numerical value (in the range [0,1]) that suggests the translation has a better quality as it gets closer to 1. BLEU is based on word overlapping and geometric mean(4.1), used together with a brevity penalty (4.2) (to avoid assigning high rewards to excessively short sentences) and computes how many n-grams from the candidate sentence match with the reference one(4.3).

$$p_n = \frac{\sum_n matched}{\sum_n count} \tag{4.1}$$

$$bp = min(1, \frac{candidate\_length}{reference\_length}) \tag{4.2}$$

$$BLEU = bp * (\prod_{i=1}^{n} p_i)^{\frac{1}{n}} \tag{4.3}$$

BLEU example

Reference: 'a', 'man', 'is', 'running'
Candidate: 'a', 'man', 'is', 'playing'
n-grams: 4

precision-1 = 3 / 4 ('a', 'man', 'is')
precision-2 = 2 / 3 ('a', 'man'), ('man', 'is')
precision-3 = 1 / 2 ('a', 'man', 'is')
precision-4 = 0
BP = 1
BLEU = 0.707

4.4.2 POS-BLEU

Part-Of-Speech BLEU [15] uses part-of.speech tags instead of words: it considers the syntax aspect of the sentence and compare how the reference and

candidate are similar in that sense.

POS-BLEU example

Reference: 'a', 'man', 'is', 'running', 'alone'
POS: 'article', 'name', 'verb', 'verb', 'adjective'
Candidate: 'a', 'man', 'is', 'playing', 'guitar'
POS: 'article', 'name', 'verb', 'verb', 'name'
n-grams: 4

precision-1 = 4 / 5 ('article', 'name', 'verb', 'verb)
precision-2 = 3 / 4 ('article', 'name'), ('name', 'verb'), ('verb', 'verb')
precision-3 = 2 / 3 ('article', 'name', 'verb'), ('name', 'verb', 'verb')
precision-4 = 1 / 2 ('article', 'name', 'verb', 'verb')
BP = 1
POS-BLEU = 0.668

4.4.3 SELF-BLEU

Lastly SELF-BLEU [16] is a metric computing the diversity of a generated text: this is a particularly important metric in a model based on GANs, since the mode collapse is one of the most diffused problems affecting them. This score is computed same as BLEU, but uses the same test as candidate and reference, and outputs a number between 0 and 1 indicating how much repetition the candidate contains.

SELF-BLEU example

Considering the document:
'a','b','c','d'.
'c', 'd', 'e', 'f'.
'a', 'b', 'b', 'c'.
n-grams: 4

precision-1 = $\frac{1.0+0.75+0.5}{3} = 0.75$
precision-2 = $\frac{1.0+0.40+0.70}{3} = 0.70$
precision-3 = $\frac{0.36+0.20+0.29}{3} = 0.28$
precision-4 = $\frac{0.26+0.16+0.22}{3} = 0.21$
SELF-BLEU = 0.42

4.5 Likert scale

Quantitative metrics such as BLEU cannot fully evaluate the quality of a text, so we decided to add another level of evaluation by assigning scores to different aspects of the sentences following the Likert scale [25]. Named after its inventor Rensis Likert, it is a scale going from 1 to (4-5) commonly used in questionnaires and surveys, with the numbers indicating how much the person answering agrees or disagrees with what has been written before. An example could be:

The problem of pollution has to be addressed as soon as possible.

- (**1**) Strongly disagree
- (**2**) Disagree
- (**3**-optional) Neither agree nor disagree
- (**4**) Agree
- (**5**) Strongly agree

This scale provides a good tool for evaluating things that are non-trivial to estimate with precision and require a human evaluator, such as generated text: with metrics it is quite hard to evaluate complex aspects of language such as coherence through the sentence or even some syntax errors. Likert scales are not perfect though, as they fall under common distortions tied to surveys and human evaluations. For example, a Likert scale without the third answer is considered to lead people to extremist views, while not perfectly representing a softer answer. Also, people are often tempted to answer in a way that appear perfect or normal and so giving an answer that differs from real identity and ideals. All of this considered, Likert scale is still a pretty good instrument of evaluation, and we used it since this problem is less prone to suffer such biases. For each sentence evaluated we adopted 3 scores from 1 to 5, divided as follows:

4.5.1 Pertinence

With pertinence we indicated how much the sentence is related to the seeds used: if a sentence is strictly related to what is presented in the seeds, then the score is high, otherwise if the context is completely different then the sentence is evaluated poorly. For example if the seed "woman tosses frisbee" generates "a woman throws a frisbee in a park.", then this sentence is evaluated 5/5,

while if it would have generated something completely unrelated as “a man is running in the gym” then the evaluation would be 1/5. As shown, this does not represents if the sentence contains all the words from a seed, but if the generic context is the same between the two elements.

4.5.2 Syntax

Syntax is made up of rules and principles that give the foundation of a language, including general sentence structure and word order. A misused verb in a specific context or general grammar mistakes could be the causes for a sentence to get low syntax score. For example, a sentence with no errors such as “a couple of people walking down the street” would get a 5 score, while if the sentence were to be “a couple of *person* walking down the street” then the syntax would be evaluated as 3 or 4, because of the misuse of the singular “person” over “people”.

4.5.3 Coherence

With coherence we wanted to evaluate how much a sentence keeps revolving around the same argument, penalizing those sentences suddenly changing the topic and so breaking the sentence in two (or more) parts completely unrelated with each other. A basic example could be the sentence “there are two dogs with their owner”, which is fine since the sentence keeps revolving around a single scenario, while another sentence such as “a dog is walking with a men that is cooking in a pan”, which although having a good syntax it lacks coherence between the first and second part.

Chapter 5

Results

In this chapter we will present the results of our experiments, together with other models used as baseline: specifically in the first part we explain some details over our experiments, followed by a confrontation between scores obtained by state-of-the-art models and ours, finishing with a human evaluation done over a set of 100 sentences selected randomly from the Image COCO dataset.

5.1 Experiments

We run our experiments over the annotations of Image COCO dataset, made up of mostly short sentences. After pre-processing the sentences (tokenization and division in batches of 32 samples with max length 20, as the average sentence is less than 11 words long), we load pre-trained GloVe embeddings of size 300 for both generator and discriminator, allowing the model to obtain general knowledge given by GloVe, which will then be modified based on our dataset local informations. At this point we start with a short pre-training for both Generator and Discriminator: the first one consists of 15 epochs, to leave the most part of the learning for the GAN part, while the second one consists of only 2 epochs, as the discriminator gets too accurate in distinguishing the data. The adversarial part runs for 50 epochs, and in each one of them, we run Monte Carlo search with roll-out policy, which is heavy on computation: considering a sentence of length l, we run the algorithm k times for each word contained in the sentence (l x k), to estimate how much the word l_i contributes to make the sentence real. Therefore, we set the k value to 8, giving us a good trade-off between time and performance. After the training process has ended, we use our test set of 6670 samples to compute all the scores showed in the next section.

5.1.1 Adam vs SGD

A special mention has to be made for the optimizers: as shown in the experimental setup, we used Stochastic Gradient Descent for both Discriminator pre-training and during GAN training for both models, while we used Adam in the Generator pre-training. This is due to what we observed during our starting tests and has been confermed thanks to [13], which specifies that adaptive optimization methods (such as adam) often find very different solution from the classic Gradient Descent (or stochastic GD in our case), showing that in test sets adaptive algorithms ofter perform a worse generalization on several learning models, but have better training performance. In our experiment we had a confirmation of this behaviour, as we observed a good training while the test performed very poorly (both with metrics and human evaluation), so switching to SGD only in GAN training allowed us to improve significantly our results. However we maintained Adam in the generator pre-training,as we observed a good behaviour and faster training.

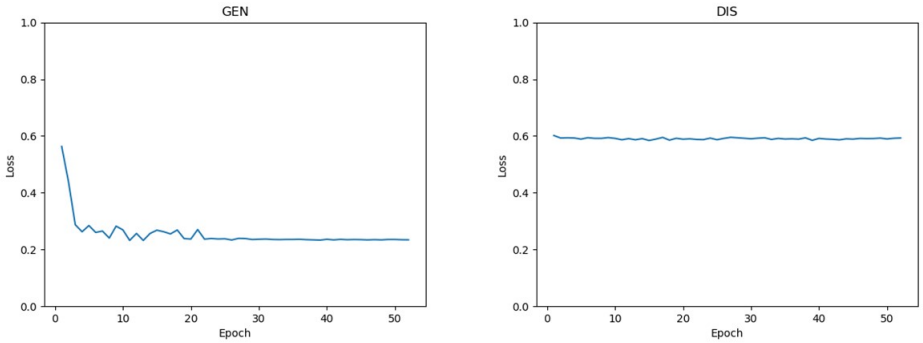

Figure 5.1: Training of the model using Adam optimizer in Generator pre-training, during GAN training and Discriminator pre-training

	BLEU	POS-BLEU	SELF-BLEU
2-gram	0.75562	0.96017	0.92066
3-gram	0.49795	0.90345	0.79215
4-gram	0.32308	0.79699	0.61746
5-gram	0.21538	0.65108	0.44389

Table 5.1: Quantitative scores considering 2-3-4-5 grams for the model trained with Adam optimizer.

As we can see, the training is very good as the loss seems to decrease until reaching a constant value, but evaluating the test set we notice a poor performance in both BLEU and POS-BLEU, indicating that the model is not capable to generate sentences that are meaningful and grammatically corrects, while using SGD we obtain far superior results as shown in the next sections.

5.1.2 Discriminator labels

During training, we found immediately a problem: the discriminator since the beginning was too strong and able to distinguish between real and fake sentences with little error, so we implemented an occasional flip of the labels used as ground-truth from the discriminator in order to slow it down. We tried this with two approaches: the first had the percentage of label swapped be constant throughout the entire training, while the other one had the error on the labels decrease over time: with the two approaches we obtained results superior to the state-of-the-art and both really comparable, so we decided to show both of them.

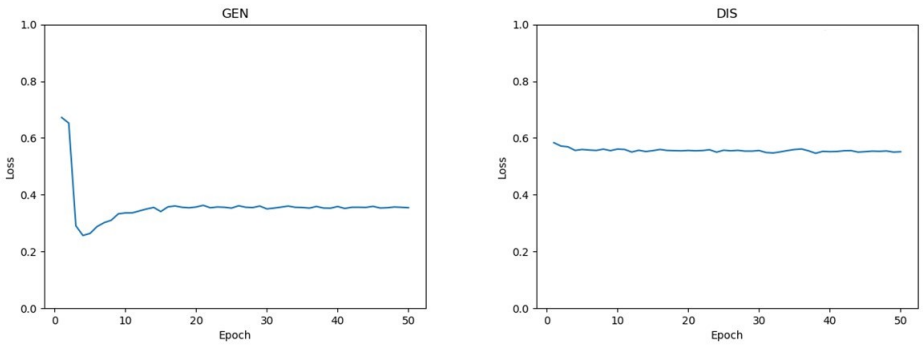

Figure 5.2: Training of 50 epochs using a constant label error (15%).

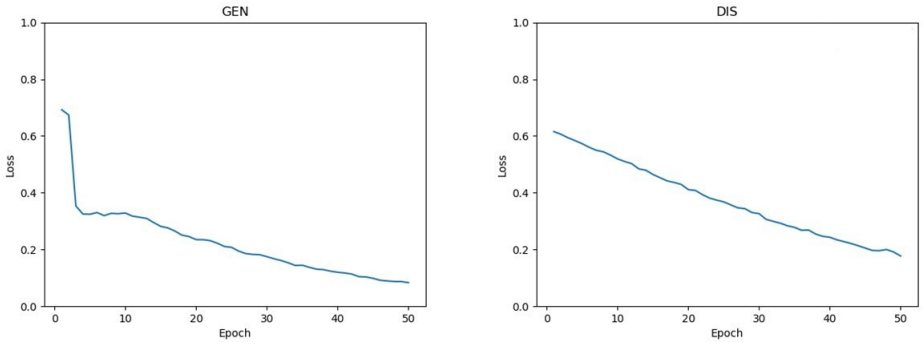

Figure 5.3: Training of 50 epochs using a decreasing label error (20% decreased at each epoch).

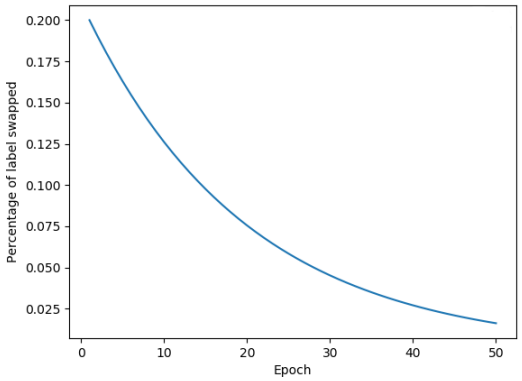

Figure 5.4: Decrease factor for the error in the labels, each value x_n is obtained by multiplying x_{n-1} by 0.95.

5.2 Quantitative results

We tested our results against the ones obtained with other famous state-of-the-art models using Texygen [23], in particular SeqGAN [4], RankGAN [18] and LeakGAN [19], all of them trained for 100 epochs. In the following table we will show the results obtained from SeqGAN, RankGAN and LeakGAN trained with the same dataset used in our work, and we will compare them with our results indicated as Tri-GAN (C/D to indicate if we used the constant label swap or the decreasing one as explained before).

	SeqGAN	RankGAN	LeakGAN	TriGAN C	TriGAN D
BLEU-2	0.76412	0.75991	0.87245	0.89369	**0.89557**
BLEU-3	0.53147	0.51714	0.74126	0.76081	**0.76457**
BLEU-4	0.33487	0.31405	0.59008	0.59922	**0.60212**
BLEU-5	0.21032	0.18971	0.43941	0.44161	**0.44403**

Table 5.2: BLEU scores considering 2-3-4-5 grams.

	SeqGAN	RankGAN	LeakGAN	TriGAN C	TriGAN D
POS-BLEU-2	0.98645	0.99147	0.99798	0.99871	**0.99885**
POS-BLEU-3	0.97587	0.97702	0.99327	**0.99539**	0.99528
POS-BLEU-4	0.94731	0.94494	0.98063	**0.98579**	0.98473
POS-BLEU-5	0.89496	0.88345	0.95847	**0.96501**	0.96289

Table 5.3: POS-BLEU scores considering 2-3-4-5 grams.

	SeqGAN	RankGAN	LeakGAN	TriGAN C	TriGAN D
SELF-BLEU-2	**0.92736**	0.94955	0.95251	0.93127	0.93386
SELF-BLEU-3	**0.78733**	0.84410	0.89048	0.83065	0.83620
SELF-BLEU-4	**0.60066**	0.69409	0.81373	0.69764	0.70627
SELF-BLEU-5	**0.42544**	0.53726	0.73119	0.55521	0.56497

Table 5.4: SELF-BLEU scores considering 2-3-4-5 grams.

We can see clearly from the results that our models performs better in both BLEU and POS-BLEU both SeqGAN and RankGAN, indicating a better structure in the sentence building, while LeakGAN is closer in scores to our results. A special mention has to be done for the SELF-BLEU which, as explained in the previous chapter, refers to the diversification of the considered text, with a lower value indicating a better variety. As shown, SeqGAN and RankGAN, in particular the former, seem to have a far better diversity in the generated text, but this is most likely a consequence of the poor quality of the sentences. A language is made up of groups of words that are more common in comparison to others, so when the generated text loses structure and overall quality, it is less likely for these words to appear as frequently as they should and so this score reflects a more chaotic choice of words rather than a real diversity in the text considered.

5.3 Qualitative results

We wanted to add another level of evaluation over state-of-the-art metrics evaluated before because, being quantitative metrics, they are not able to represent some qualitative aspects contained within natural language. All of this considered, we decided to run an experiment taking 100 samples randomly selected from our test set, and evaluated them over the different aspects of pertinence, syntax and coherence as described before. We run this experiment for both our models, constant and decreasing, and obtained the following results:

	TriGAN C	TriGAN D
Pertinence	3.340	**3.510**
Syntax	**3.945**	3.935
Coherence	3.485	**3.790**

Table 5.5: Human evaluation done over some test sentences.

As we can see the two models are quite close to each other, with the decreasing model performing slightly better in syntax and coherence, meaning that the decreasing label error helps the model in building a better sentence structure and maintaining the coherence throughout the whole generation, although we have very short sentences in our dataset. The constant model performs a bit better in pertinence, meaning that the topic of the generated sentences are closer to the one expressed in the seeds.

5.3.1 Discussion

When evaluating the sentences we need to consider that, given the nature of the dataset pre-processing, some seeds could make less sense than others: when we extract two nouns and a verb there is the chance that the choice of these elements could be non-optimal, therefore giving birth to meaningless seeds. This is because there is no method to pick exactly the structure subject-action-object inside a sentence, but the pick is in order inside the sentence from left to right. This simplifies the pre-processing but at the same time generates seeds that are hard to expand and so generating bad samples. For example considering the sentence “A man with a sweater is riding a snowboard.” should give the seeds “man riding snowboard”, while instead it gives “man is snowboard”. This sample shows a choice of seeds clearly non-optimal, and also brings up another problem within the pre-processing: some verbal forms such as present continuous are “truncated”

to the first part of the verb, reducing or even disrupting the context of the sentence itself. We did not find effective solutions to these problem, which are intrinsically problems of evaluating a language in a quantitative way: smoothing these problems could improve drastically the performances in our model.

Chapter 6

Conclusions

In our work we proposed a new approach for text generation starting from an acquired knowledge that builds a general context starting from GloVe and then specializes it with informations obtained from the training dataset. The seeds are then used to focus on a limited part of this context and build a sentence over that specific topic, allowing to generate new and every time different sentences. Using the SeqGAN model [4] as inspiration, we built a Generative Adversarial Network model using as Generator a sequence-to-sequence model and as Discriminator an attention based Long Short-Term Memory. The generator had the task of taking 3-word seeds and generating a full length sentence, having as main topic the same as the original seed. To do this we used an encoder (modeled over a Gated Recurrent Unit) to compress the seeds and obtain a local context, and then expanded it with the decoder (again a Gated Recurrent Unit). To train this GAN model we draw heavily from the REINFORCE algorithm already used in the SeqGAN, although with some modifications, that allowed us to overcome the differentiability problem of discrete data.
From the results we notice that the model can produce realistic text in this specific task using short sentences, but it produced also samples that are either totally disconnected from the original context and/or structurally wrong, for example having words repeated two or more times. From the quantitative metrics we can tell that our model obtained results superior to the state of the art, although the room for improvement is still huge. Considering also the qualitative evaluation, the model stays around the medium-upper part of the likert scale, which hints us that we have overall a discrete quality but with some critical samples that bring down the results.

6.1 Future work

Although we obtained quite good results if compared with the state of the art, several improvements come to mind when thinking of the experiments: first of all a finer tuning of the hyper parameters could bring a lot of improvements, as we were not able to try all the possible combinations. Also, using different types of discriminators, for example a self-attentive discriminator [24] or one based on Convolutional neural network [5] could lead to other results. Another interesting improvement could be implementing the leak system from [19], which could step up our syntax and coherence over long sentences. Again, improving the pre-processing would allow us to eliminate meaningless seeds and help building a more solid sentence structure. Lastly, an important experiment to run would be to try change the seed structure, considering for example sentences with 4, 5 or more seeds, maybe a variable number for each sentence. Leaving behind the noun-verb-noun paradigm could also give new perspectives, therefore considering only the most relevant words in general, maybe obtained with an attention net during the pre-processing of the dataset.
With this and other possible improvements there are countless applications for such model, for example chatbots, where we could extract seeds from a question given by the user and therefore produce a proper response, or even systems for generating human readable text and write essays of articles over various topics.

Bibliography

[1] Hochreiter S., Schmidhuber J., LONG SHORT-TERM MEMORY, Neural Computation 9(8):1735-1780, 1997

[2] Cho, et al., Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation (2014), arXiv:1406.1078

[3] Goodfellow, Ian and Pouget-Abadie, Jean and Mirza, Mehdi and Xu, Bing and Warde-Farley, David and Ozair, Sherjil and Courville, Aaron and Bengio, Yoshua, Generative Adversarial Nets, Advances in Neural Information Processing Systems 27:2672-2680, 2014

[4] Lantao Yu, Weinan Zhang, Jun Wang , Yong Yu, SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient, arXiv:1609.05473v6, 2017

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

[6] Turing A.M., Computing machinery and intelligence. Mind, 59, 433-460., 1950

[7] Ilya Sutskever, Oriol Vinyals, Quoc V. Le, Sequence to Sequence Learning with Neural Networks (2014), arXiv:1409.3215v3

[8] Rohit Prabhavalkar, Kanishka Rao, Tara N. Sainath, Bo Li, Leif Johnson, Navdeep Jaitly, A Comparison of Sequence-to-Sequence Models for Speech Recognition, (2017) https://www.isca-speech.org/archive/Interspeech_2017/pdfs/0233.PDF.

[9] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need, (2017), https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf

[10] 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).

[11] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollar, Microsoft COCO: Common Objects in Context, arXiv:1405.0312v3 [cs.CV] 21 Feb 2015

[12] Jeffrey Pennington, Richard Socher, Christopher D. Manning, GloVe: Global Vectors for Word Representation, aclweb.org/anthology/D14-1162

[13] Ashia C. Wilson, Rebecca Roelofs, Mitchell Stern, Nathan Srebroy, and Benjamin Recht, The Marginal Value of Adaptive Gradient Methods in Machine Learning, arXiv:1705.08292v2 [stat.ML] 22 May 2018

[14] Kishore Papineni, Salim Roukos, Todd Ward, Wei-Jing Zhu, BLEU: a Method for Automatic Evaluation of Machine Translation, https://www.aclweb.org/anthology/P02-1040

[15] Maja Popovic, Hermann Ney, Syntax-oriented evaluation measures for machine translation output, https://www.aclweb.org/anthology/W09-0402

[16] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, Yong Yu, Texygen: A Benchmarking Platform for Text Generation Models, arXiv:1802.01886v1 [cs.CL] 6 Feb 2018

[17] Tong Che, Yanran Li, Ruixiang Zhang, R Devon Hjelm, Wenjie Li, Yangqiu Song, Yoshua Bengio, Maximum-Likelihood Augmented Discrete Generative Adversarial Networks, arXiv:1702.07983v1 [cs.AI] 26 Feb 2017

[18] Kevin Lin, Dianqi Li, Xiaodong He, Zhengyou Zhang, Ming-Ting Sun, Adversarial Ranking for Language Generation, arXiv:1705.11001v3 [cs.CL] 16 Apr 2018

[19] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, Jun Wang, Long Text Generation via Adversarial Training with Leaked Information, arXiv:1709.08624v2 [cs.CL] 8 Dec 2017

[20] Yizhe Zhang, Zhe Gan, Kai Fan, Zhi Chen, Ricardo Henao, Dinghan Shen, Lawrence Carin, Adversarial Feature Matching for Text Generation, arXiv:1706.03850v3 [stat.ML] 18 Nov 2017

[21] William Fedus, Ian Goodfellow, Andrew M. Dai, MaskGAN: Better Text Generation via Filling in the_____, arXiv:1801.07736v3 [stat.ML] 1 Mar 2018

[22] Massimo Caccia, Lucas Caccia, William Fedus, Hugo Larochelle, Joelle Pineau, Laurent Charlin, Language GANs Falling Short, arXiv:1811.02549v5 [cs.CL] 17 Aug 2019

[23] Yaoming Zhu, Sidi Lu, Lei Zheng, Jiaxian Guo, Weinan Zhang, Jun Wang, Yong Yu, Texygen: A Benchmarking Platform for Text Generation Models, arXiv:1802.01886v1 [cs.CL] 6 Feb 2018

[24] 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).

[25] Likert,Rensis (1932). "A technique for the Measurement of Attitudes". Archives of Psychology 140: 1-55.