University of West Bohemia
Faculty of Applied Sciences
Department of Computer Science and Engineering

Bachelor's thesis

Neural networks for generating movie dialogues.

Pilsen 2020 Mukanova Zhanel

Místo této strany bude zadání práce.

Declaration

I hereby declare that this bachelor's thesis is completely my own work and that I used only the cited sources.

7th May 2020

Mukanova Zhanel

Acknowledgment

I want to thank all my teachers who taught me in my bachelor degree. Also, I want to thank my supervisor Ing. Miloslav Konopík Ph.D. for his advice in my bachelor thesis.

Abstract

Natural language generation (NLG) is an area located at the intersection of artificial intelligence and computational linguistics. The goal is to produce written or spoken narrative from a structured dataset.

NLG is one of the most important technology in artificial intelligence. However, a complete understanding and using the meaning of the language is an extremely difficult task considering the human language has many sophisticated features.

This paper is an overview of the machine learning method's effectivity for text generation. This work aims to analyze existing algorithms and methods of artificial neural networks used to solve this problem. And to demonstrate how NLG software works, I created a web application with trained models.

Abstrakt

Generování přirozeného jazyka (NLG) je oblast, která se nachází na křižovatce umělé inteligence a počítačové lingvistiky. Cílem je vytvořit písemný nebo mluvený příběh ze strukturovaného souboru dat.

NLG je jednou z nejdůležitějších technologií umělé inteligence. Úplné porozumění a používání významu jazyka je však nesmírně obtížný úkol vzhledem k tomu, že lidský jazyk má mnoho složitých funkcí.

Tato práce je přehledem účinnosti metody strojového učení pro generování textu. Cílem studií je analyzovat stávající algoritmy umělých neuronových sítí používaných k řešení tohoto problému. A abychom ukázala, jak funguje software NLG, vytvořila jsem webovou aplikaci s vyškolenými modely.

Contents

1. Introduction

Natural Language Generation (NLG) is a subsection of Natural Language Processing (NLP) [Yse20]. NLG is a fast-moving field of Machine Learning and Artificial Intelligence. This system turns structured data into a written narrative, while NLP processes turn text into structured data.

The first attempt to automatically generate fiction was made by the French writer Raymond Queneau, published in 1961. The writer wrote the book "Hundred Thousand Billion Poems". This book contains ten sonnets with 14 lines, which was written in strips, due to this another can replace any line. As follows that accurately 10^{14} different sonnets may be produced using this book. It was the first combinatoric try to text generation [Dow02].

Today the term Natural Language Generation has become more and more common. NLG is used to automate the writing of reports or product descriptions. With this system, thousands of unique text is in less time can be produced than somebody would write them manually.

The goal of NLP is to communicate with predicting the next word in a sentence. The problem is which word from millions of possibilities we can predict. We can solve it by using Language models, which can be constructed by at a character level, word level and sentence level. Neural networks such as **RNNs**, **LSTMs** and **GANs** have allowed processing of long sentences with good accuracy of language models, which estimates the relative likelihood of different phrases [Osb19].

Machine learning methods, especially for networks with GAN architecture, are mainly discussed in the thesis.

1.1 Motivation

The goal of this thesis is to explore the current state of text generation and implement RNN and GAN architecture for the film dialogues generation. The dialogues are needed to be readable with less emphasis on understandably.

In this project, an understanding of the neural networks will be got within automatic text generation and also a web application will be created to demonstrate how text generation works. We will look at how the text was generated using different machine learning methods and then try to attempt this information into utilizing a GAN to create film dialogues.

In this work, several different architectures will be used, evaluated and finally, used the ideas we have received to create a better model. The original GAN for generating text will be separately considered. Different models will be trained in different settings and architectures. And then they will be evaluated using perplexity and human evaluation.

1.2 Goals

In this thesis, we are going to investigate the principle of Neural networks for text generation. As stated in the Motivation, the main aim is to implement an NLG system based on GAN architecture. First of all, understanding what is Neural Networks mean will be got. Then we will discuss what kind of NN architectures exist. In the last step, some text generating models will be examined in more detail. All general information is described in the Problem Statement chapter.

Before implementing the NLG system, training data will be prepared in XML format. This format must include the name of the film, year, genre and also subtitles. All training data saved in utf-8 encoding. As for subtitles databases, several servers will be used: **opensubtitles.org**, **subscene.com**, **subtitles.cz**, **yifysubtitles.com** and **podnapisi.net**.

The next step is to implement an NLG system based on GAN and RNN. The models will be trained in different settings and architectures and evaluate them.

At the last step, a web application will be created to demonstrate the work of the NLG system.

2. Problem Statement

In this chapter, language models that based on neural networks such as *Recurrent neural network*, *Convolutional neural network* and *Generative Adversarial Networks* will be discussed.

2.1 Text Generation

The main goal of Natural Language Generation (NLG) can be defined as producing new universally understandable texts in human languages from some non-linguistic or linguistic information representation. NLG is computational linguistics and artificial intelligence subfield.

Exist two methods to text generation data-to-text and text-to-text, that are instances of NLG. Text-to-text generation methods take as input existing text and generate new understandable text. However, sometimes it is needed to generate text from non-linguistic input such as spreadsheets or database records. Data-To-Text generation methods used for these needs [Gat18].

Also NLG can be split into the *non-linguistic task* and the *linguistic task*. Content determination and text structuring are defined in non-linguistic subtask.

Ehud Reiter and Robert Dale [ER97] describing linguistic NLG problem like 6 subproblems.

- **Content determination -** this task, is decided what information should be included in the text.
- **Discourse planning -** determining the order of information in a sentence.
- **Sentence aggregation -** grouping several messages into sentences.
- **Lexicalization -** find relations between words and phrases.
- **Referring expression generation -** is the task of selecting words or phrases to identify domain entities [ER97].

- **Linguistic realisation -** task, that using grammar and language rules to construct correct sentences.

In the last task called Linguistic realisation, all processed words and sentences transform into a coherent natural language. All syntactical, morphological, and other rules should be applied on output text, but they may not be included in input data. Exist several solutions to solve this problem which is described below.

Template-based models have many benefits. For example, their output is always grammatically correct and do not contain generation errors. However, on the other hand, this model has several disadvantages. On account of the fact that templates need to be constructed manually, the system should require much time and human resources. This model is not flexible, which means that templates can not be used in different contexts, and they are not automatically extendable. Moreover, the most important thing that this model is not able to learn.

Handcrafted rule-based models are more advanced than template-based models, but still expect human resources and time to generate sentences. This model using a set of human-created rules that define a mathematical model. The rule-based models are knowledge-intensive. They take long time and expert knowledge and feedback to be developed. Moreover, rule-based models are usually able to generate high-quality natural language text.

Statistical models are based on the likelihood of occurrence of a word. It geared towards the previous sequence of words that were used in the text and take less human effort. Statistical language model does not require any semantic relational knowledge and takes much less time to generate output text [JM16].

2.2 Neural Networks

The neural network or artificial neural network is one of the most popular machine learning algorithms at present. The concept of the artificial neural network was inspired by the neural architecture of a human brain. The functionality of artificial neuron is similar to a human neuron, it takes in some inputs and creates an output [dee20].

A neuron in machine learning contains a mathematical function that termed as an activation function. The most popular activation functions are sigmoid, tanh, ReLU and softmax [Nig18]. The idea behind artificial neuron is that it is possible to mimic certain parts of neurons, such as dendrites,

cell bodies and axons using simplified mathematical models. Signals can be received from dendrites and sent down the axon once enough signals were received. This outgoing signal can then be used as another input for other neurons, repeating the process [Nag18]. Thus, neurons are gathered into layers of a neural network.

Any neural network has one input, one output layer and a required number of hidden layers. This number of hidden layers depends upon the complexity of the problem to be solved. Moreover, each of the hidden layers can have a different activation function which depends on the problem in question and the type of data being used.

Neurons learn certain weights at every layer to make a prediction. The algorithm through which they learn the weights is called *backpropagation*. A neural network that have more than one hidden layer is generally called a Deep Neural Network [Nig18].

2.2.1 Convolutional neural networks - CNN

A **Convolutional neural network** is basically a neural-based approach, that is applied on matrix. The word *"convolution"*n, explains that the basic operation in this network is the convolution operation. The CNN contains one or more convolutional layers, pooling or fully connected. These layers (Figure 2.1) calculate the mathematical operation between a part of input and *kernel*. The final result of this process of the convolutional operation is called a *feature map*. The parts are defined by the size of matrix, step size, variant of moving kernel over the input and else [IG16].

The Convolutional Layer (The Kernel/Filter)

The Convolutional Layer is an element involved in the convolution operation in the first part of the convolutional layer. It contains a set of independent filters that are randomly initialized. This filters can detect low-level features [Sah18].

Pooling Layer

The Pooling Layer is responsible for reducing the matrix size of the Convolved Feature. This layer should reduce the processing power needed to process the data by reducing the dimensionality. Many pooling types exist, but most common are Max Pooling and Average Pooling [Sah18].

- **Max Pooling** returns the maximum value. It can be used as a Noise Suppressant.

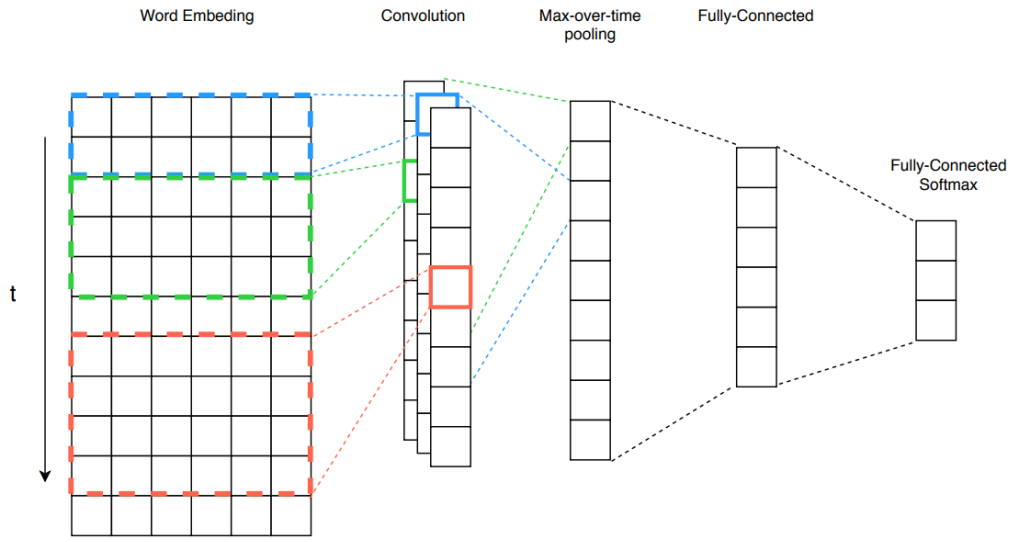

Figure 2.1: An illustration of basic CNN model for text classification taken from the [SR19]

- **Average Pooling** returns the average of all the values.

The main goal of Pooling layer is to reduce the amount of information in each feature obtained at the convolutional level, leaving only the most essential information.

Classification (Fully Connected Layer)

The fully connected Layer on Figure 2.1 takes the output of previous layers and flattens them, then turns them into one vector, which can be input for the next stage. The softmax activation function, which is on the last fully connected layer, is normalizing the output into probabilities. The fully-connected layer with softmax is used to get the final probabilities.

Natural Language Processing (NLP) can be implemented with Deep Convolutional neural network (DCNN) when the output of one layer is fed to the next layer [Mic17].

2.2.2 Recurrent neural network - RNN

A Recurrent neural network is one of the classes of artificial neural networks with a linear recursive structure. In contrast with CNN, that specialized in processing with grid-like data. The RNN owning to the linear recursive structure can work with sequential data with different length. The

basic idea of RNN is to share parameter across a deep computational graph. At each step, RNN takes part of the input and produces a function of the output, that is used in the next step to prepare the next output. This recurrent formulation makes it attainable to use similar weights over completely different positions in time [IG16].

Often two architecture of recurrent unit are used: the *Long Short Term Memory (LSTM)* and the *Gated Recurrent Unit (GRU)*. LSTM is described in the next section.

In conclusion, when RNN process sequences of tokens, it keeps track of state that represents the memory of the previous tokens. This ability makes RNN very useful in language processing.

Although an RNN is a simple and powerful model, in practice, it is difficult to train. There are many learning algorithms for this model, such as Backpropagation Through Time (BPTT), Real-time Recurrent Learning and others. Most of these approaches are based on gradients, which have little success in solving complex problems [Nab19]. The main reasons why this model is so unwieldy are the vanishing gradient and exploding gradient problems described in Bengio et al. (1994) [BSF94].

Long-short-term-memory (LSTM)

The LSTM is one of RNN architecture, that is used in the deep learning field. This variation was introduced by German researchers Sepp Hochreiter and Juergen Schmidhuber [SH97]. The main idea of LSTM is to solve the vanishing gradient problem partially, that can be backpropagated through time and layers.

The goal of LSTM is to remember the information for a long period of time. This specificity is very suitable or processing, predicting and classifying data in time. Owning to this fact LSTM can predict the next word after a big distance between dependencies in sequences.

A typical architecture is consist of a cell and three neural gates (input, forget and output). The cell is representing a memory part of the LSTM unit, that chooses what information it needs to store and then it allows to read, write or erasures by gates that open and close.

2.3 Generative Adversarial Network (GAN)

In this chapter, a generative adversarial network is explained and discuss possibilities to solve some problems in text generation.

2.3.1 Introduction

The GAN are a new facility in machine learning that has attained great results in generating realistic synthetic data. GAN belongs to generative models, that process data according to the principle of maximum likelihood. The main goal of maximum likelihood in generative models is to select some parameters for the model that will increase the likelihood of the training set.

Ian Goodfellow [Goo17] introduced GAN like a probability model. We can understand GAN like a system for training deep neural networks using a minimax game. Learning in this game is equivalent to minimising Jensen-Shannon divergence between the model and the data distributions. Two players that are depicted like two functions. Each of them has his input data and own parameters. The first player is the Generator, that is represented like function **G**. He takes input data **z** and parameters $\theta^{(G)}$. The second player is Discriminator, that defined by function **D**. Is differentiable with respect to parameters $\theta^{(D)}$ and input data **x**. All players have cost functions, that depends on parameters from another player. Moreover, players have no access to other's player parameters. The Generator tries to maximise the final classification error between fake and real data, while Discriminator is trained to minimise the final classification error (Figure 2.2).

One an essential advantage of generative models is that they can be trained with missing data and can provide predictions on inputs with missing data. Often, machine learning algorithms are using for generating a raft of labelled training data to be able to generalize well. Despite this, *semi-supervised* learning is one strategy for reducing the number of labels. The learning algorithm can improve its generalization by studying a large number of unlabeled examples which, which are usually easier to obtain [Goo17]. The *semi-supervised* learning strategy can be used in generative models.

Next advantage of GAN is possibility machine learning to process *multi-modal* outputs. For explaining, it is mean that each input may correspond to multiple different outputs, which are all correct. Traditional training models of machine learning are based on minimizing the differences, such as the mean squared error between the aspired output and target, are not enabled to train models, which works with multi-modal outputs.

2.3.2 Generator

The Generator takes the random vector z from a latent vector space as input and generates a sample G(z) that is similar to real data, which were used in training. This vector is drawn from a Gaussian distribution, and

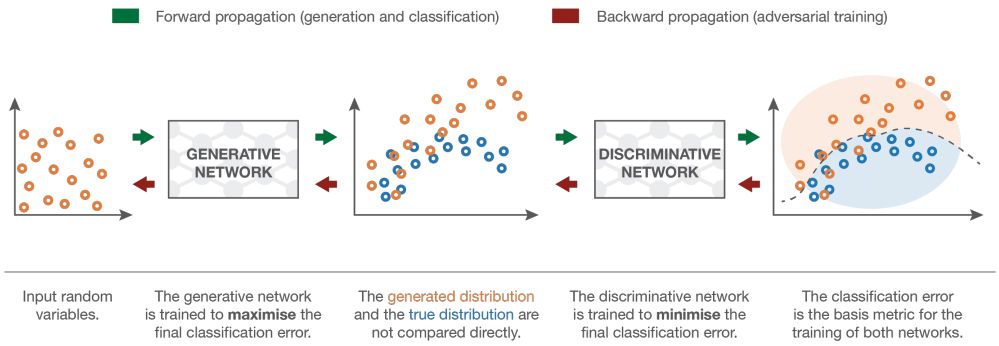

Figure 2.2: An illustration of GAN structure example taken from the [Rocay]

it is able to use as a seed in the generation process. After the training process, a generator function is saved and used to generate new samples, which are fed to the Discriminator function D. The main goal of Generator is to generate samples same as real data from random noise and maximise the final classification error between fake and real data. Naive Bayes, mixtures of multinomials, mixtures of Gaussians, HMM, bayesian networks algorithms can be used as a generator function [Bro19].

2.3.3 Descriminator

Discriminator can be implemented as a classification algorithm, that trying to predict one of two classes (real or fake) and minimise the final classification error. For training, Discriminator uses supervised learning techniques, that gives a high likelihood to real data samples and a low to samples generated by the generator [Goo17].

The Discriminator can be implemented as any classifier like *Support Vector Machines*, *Nearest Neighbor* and others.

2.3.4 GAN in text

The Generative Adversarial Network (GAN) has reached great heights in generating realistic synthetic data. However, convergence problems and difficulties associated with discrete data discourage using GAN in text. The main issue is that the discrete data after the generative model can't easily get through the gradient update between discriminative model and the generative model (Figure 3.3). Another problem is that the discriminative model can estimate only complete sequence. Because in partial chains, it is

hard to balance current and future score once the whole sequence has been generated [LY17].

2.4 Text Representation

Representation of text is essential for the performance of many real-world applications. The goal is to turn language into something that a computer can process. Normally, processors work with simple arithmetic like an adding or multiplying numbers. Language models typically use probabilities or frequencies to character-level [Kar15], on word-level [ZC16] as well as sentence-level and even on document-level [LM14].

2.4.1 Character level

As a character-level input, the original sentence is decomposed into a sequence of characters, including special characters. As output, it produces a small vocabulary, like as alphabet [DL17]. Due to the small number of elements in the dictionary, this model is quicker to train than other language models. The character-level model requires less memory, whereby it has fast processing. In large vocabularies, it becomes very ineffective because it drops the relationships between words.

Regularly, characters are encoded by the one-hot encoding model, that is effective to small vocabularies.

2.4.2 Word Sequences language model

Statistical language models are a type of model that gives probabilities to word sequences. N-gram is the simplest language model. N-gram can be understood as the sequence of N words.

Regularly word-level models show higher accuracy than character-level language models. The model can predict the probability of the next word in a sequence based on words already existing in the sequence, that helps to save relationships between words [Nel19]. The disadvantage is that it requires more memory due to the extensive vocabulary.

2.4.3 Encodings

As described earlier, for preprocessing, it is important to prepare the incoming data in the way that the computer can understand it. Two main

encoding methods *One-Hot Encoding* and *Word Embeddings* are described in these subsections.

One-Hot Encoding

For a good understanding of one-hot encoding, a categorical variable term is needed to describe. A categorical variable is a variable that can be easily assigned to a certain category, and at the same time, variables can not be clearly sorted inside the category. For example, hair colour is a categorical variable, that having several categories: blonde, brown, brunette, red, etc. There is no definite way to sort them from highest to lowest [fDRE20].

The problem with categorical data is that most machine learning models need all input and output variables in numerical form. One-hot encoding is used to solving this problem. [Rah19].

A one-hot encoding is a representation of categorical variables as binary vectors [Bro17]. The number of categories determines the length of these vectors. For one-hot encoding is required to convert all categorical values into integer values (Figure 2.3). Then each integer value (categorical values) is represented as a zero binary vector where the index of the integer set to 1.

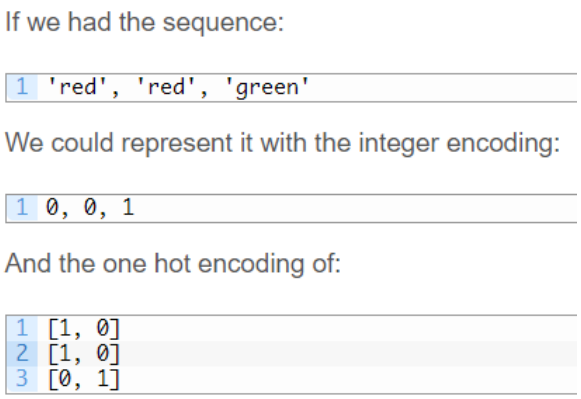

Figure 2.3: An illustration of one-hot encoding example taken from [Bro17]

Word Embeddings

Word embedding is quite similar to one-hot encoding. However, this method can use more numbers than 0 and 1 (for example, 0.1, 0.6), so word embedding can form more complex presentation forms. These representations, in contrast to one-hot encoding, can collect information about the relationship

of words, context, morphological signs, etc. Word embeddings can group (Figure 2.4) because semantically similar words have similar vectors. These vectors take a similar area of the matrix, that helps catch context and semantics [Nel19]. The most popular methods of word embeddings are GloVe and Word2Vec.

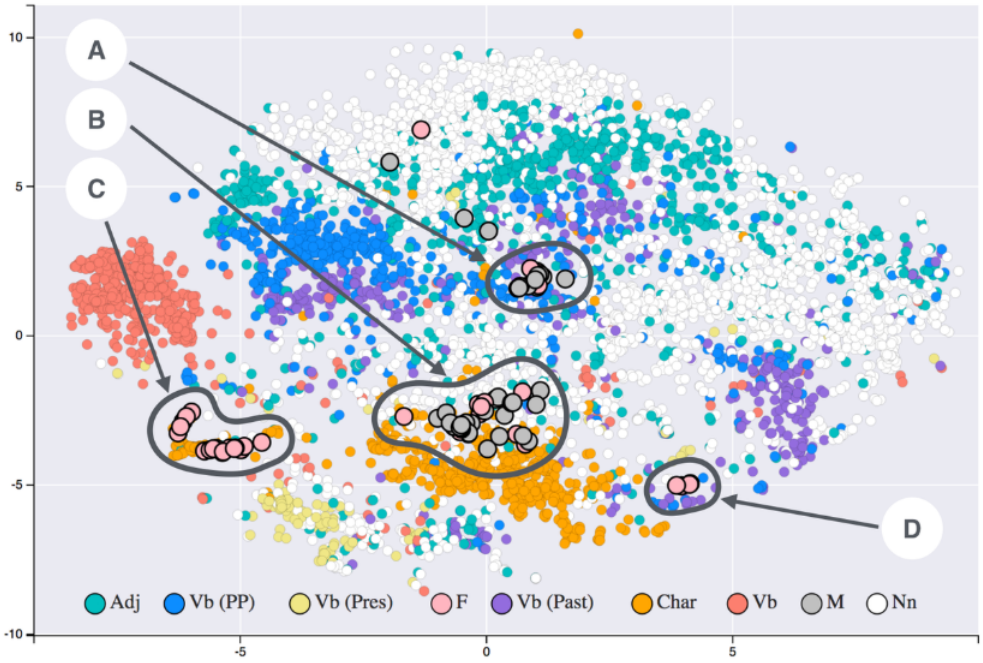

Figure 2.4: An illustration of Word Embeddings representation example taken from [Nel19]

3. Implementation

3.1 Data-set preparation

Due to the fact that each model uses different input data formats for text generation, it is necessary to prepare the basis from which it was possible to form the desired format later. The process of creating a basis of training data is described in this section.

3.1.1 Subtitles sources

The first step in preparing the training data was to decide which servers will be used for subtitles downloading. After several samples of automatic subtitle download, the following problems were revealed:

- Paid servers
- Limit on downloading subtitles per day
- Advertising

The problem with advertising and the limit was partially solved by purchasing a paid account. The download limit remained but was increased significantly from 3 subtitles to 1000 per day.

After another attempt to automatically download from servers, another problem emerged - it was IP blocking. It came because the program began to download subtitles too often and in a small period of time, thereby simulating a dos-attack. This issue was avoided with VPN and setting of "politeness interval" after each download. The download time has increased significantly, but IP was no longer blocked. After analyzing all available servers with Czech subtitles, the following servers were selected:

- https://www.opensubtitles.org/
- https://www.podnapisi.net/

- https://subscene.com/
- http://www.subtitles.cz/
- https://www.yifysubtitles.com/

Other servers were sufficiently protected from the automatic download.

3.1.2 Implementation and issues

The main task of this script was to extract all the necessary information from the subtitles server, download and convert subtitles. For this purpose, several sub-tasks were identified:

1. **Web scraping**: extract the data such as movie name, genre, year and download link from all web servers.
2. **Formating**: download subtitles and convert them to a single xml format with utf-8 encoding.

Web scraping issues

The main problem in web scarping was the design of different web pages. Writing one script to extract data from different pages was difficult because each server had its own structure. Because of this, it was decided to make several scripts for each site that could collect data and subtitles. The following objects were created:

- `opensubtitles.py`
- `podnapisi.py`
- `subscene.py`
- `subtitles.py`
- `yifysubtitles.py`

Each script contains constants:

- `RESULT_FOLDER_NAME` - path to save the result XML file.
- `BASE_URL` - web site base URL.
- `MAX_OFFSET` or `PAGE_COUNT` - the maximum number of pages on the site.

All objects contain function `get_list_of_films_from[server name]()`, which finds the necessary data on the page, saves them and combines them in one XML file.

The next problem was using sessions. Some servers required registration for free download, which complicate downloading subtitles programmatically. The solution was to use the `requests` module with cookies. Cookies are not eternal object, and they sometimes needed to be updated, but for disposable subtitles downloading, this solution is optimal.

Formating issues

With the formatting subtask the following problems appeared:

1. Sometimes when downloading subtitles in ".srt" format, there was a different format inside (for example ".sub"). Because of that, the library `pysrt` could not read the file and threw an exception.

2. Downloaded subtitles were in different encodings.

To solve the first problem with the formats was to create a function that tried to determine the format of the downloaded file. Files with the extension ".txt" and sub were converted to ".srt" format. If the downloaded ".srt" file had a different format inside, then the function tried to convert this file to the correct ".srt" format. If the file could not be fixed after all conversion attempts, the function skipped this file.

The second problem was solved by using the Linux program - `enca`. The file with the correct ".srt" format was converted to utf-8 encoding for further conversion to XML.

3.1.3 Structure

```
bachelor
|_ convert_sub_to_srt.py
|_ imdb_parser.py
|_ make_format.py
|_ subtitles_downloader.py
|_ czech_servers
   |_ opensubtitles.py
   |_ podnapisi.py
   |_ subscene.py
   |_ subtitles.py
   |_ yifysubtitles.py
```

Listing 3.1: Project root directory structure

Project root folder

Project root directory (Listing 3.1) includes objects for making correct XML format.

- **convert_sub_to_srt.py** - is the script for converting ".sub" format into ".srt" format.
- **imdb_parser.py** - contains helper class Movie, which is used for saving subtitles information and subtitles. Also contains functions for find films in IMDB database and extract genres if exists.
- **make_format.py** - is the script, that contains functions for subtitles downloading, extracting from ZIP or RAR, encoding and save result file into XML.
- **subtitles_downloader.py** - contains the main function.

Czech_servers folder

This folder contains scripts for extracting data from servers. All of these objects contain page count, save folder path constants and function for extract data. Folder includes these scripts:

- **opensubtitles.py** - The server www.opensubtitles.org requires the purchase of paid access, because of this, cookies are needed to be updated before starting.
- **podnapisi.py**
- **subscene.py**
- **subtitles.py** - Server www.subtitles.cz is free, but requires to be registered, because of this, cookies are needed to be updated before starting.
- **yifysubtitles.py**

3.1.4 Requirement specification

Programming language

As the programming language for this task was chosen a Python. The main reason for choosing Python was that it can be used in many programming tasks such as scripting tasks, web development, and math problems. For

this programming language, a large number of libraries have been created. Some libraries greatly simplify and speed up the work. Another important reason was that Python runs on an interpreter system. Due to the fact that code can be executed as soon as it is written, working with it can be very fast. [w3s20]

requests

The most used library in this project is `requests`. It helps to send HTTP/1.1 requests without any query strings. Basic HTTP methods, such as GET and POST, determine what action is tried to perform while making an HTTP request. Other well-known methods are not needed in this project. One of the essential HTTP methods in our problem is GET, that gets or retrieves data from a resource. Requests library also allows using cookies and session, that is too important in this project. [Ron20]

BeautifulSoup

BeautifulSoup is a Python library that quickly helps to scrap information from a web-site. It is one of the most popular Python parsers such as lxml and html5lib. It also allows to automatically convert incoming documents to Unicode, and outgoing documents to UTF-8. This library, allows to search, iterate the parse tree quickly. This library was chosen because of these powerful things. The information (movie name, genres and year) from webs were collected via BeautifulSoup. Beautifulsoup4 4.8.2 was used in this project. [Ric19]

RarFile and ZipFile

ZipFile [Klu16] and RarFile [Kre19] are Python modules archive reading. They both have fame interfaces. In this project, these modules used to extract subtitles from the archive. The most important thing why these modules were chosen is that they support Unicode filenames.

enca

Enca (Extremely Naive Charset Analyzer) [DN09] is Linux terminal program, that recognizes the encoding of text files. It can convert text to another encoding without knowing the original encoding. It supports most languages of Central and Eastern Europe and several Unicode variants independent of language. This program is used in the project because it solves the problem with different subtitles encoding.

pysrt

Pysrt is a Python library used to edit, modify or create SubRip files [Bou20]. In this project, it is used for simply parsing SubRip files and convert them to XML format.

xml.etree.ElementTree

The ElementTree is a library that allows working with XML files. This library is a flexible container object, designed to store hierarchical data structures in memory. The type can be described as a cross between a list and a dictionary [Fou20c].

3.2 Text generation using RNN.

The main objective of this work is to use personal trainer data to generate film dialogues. Models RNN and GAN were chosen as the basis of this task. However, the implementations of this type of architectures are non-trivial and would require much time to create and optimize it. Because of this, it was decided to use existing implementations from open sources.

3.2.1 General information

The first task was to try to use an existing RNN example code to generate text. The example [cit20b] that was selected is located on the Google Colaboratory service.

This example demonstrates how to generate text using a character-based RNN. It works with a dataset of Shakespeare's writing from Andrej Karpathy's The Unreasonable Effectiveness of Recurrent Neural Networks. Given a sequence of characters from this data ("Shakespear"), train a model to predict the next character in the sequence ("e"). Longer sequences of text can be generated by calling the model repeatedly [cit20b].

All code is implemented using TensorFlow framework [cit20b] and GPU.

3.2.2 Input data-set preparation

A training data-set was prepared before the start. Earlier in section 3.1, the process of creating XML files containing subtitles and basic information about them has been described. However, for this example, it is needed to use clear text without any markers (hour). For this task, a quick and small

script was created to extract subtitles from XML format and to create a single text file. The resulting file has the following characteristics:

Language:	Czech
File type:	text file
File size:	127MB
Unique tokens:	447
Sentences number:	5029493

Table 3.1: RNN training input file information.

3.2.3 Implementation

An example from Tensorflow website is based on a character-level method to process text. After reading a training file, the example script calculates a number of unique characters. Further, each character is mapped by a unique number, and two tables are created: one mapping characters to numbers, and another for numbers to characters.

For better and quick results several parameters in `rnn_generation.ipynb` were modified:

- `seq_length` is 60 - the maximum length of one sentence
- `BATCH_SIZE` is 256 - indicates the number of training examples used in one iteration
- `EPOCHS` is 8 - the number of epochs

When `EPOCHS` is set on more than 8, the model was "overtrained" and the result was unreadable. The result of overtrained RNN model can be seen in Figure 3.1.

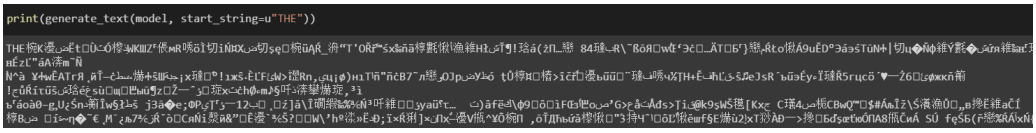

Figure 3.1: An illustration of overtrained model of text generation using RNN

After 5380 seconds of RNN training using GPU, checkpoints files were created. This files will be later used in text generation. The result of the first attempt to generate text from subtitles is shown in Figure 3.2.

THERV je pro všechny nemoce,
osobní letadlo.....
- To jsou deset prací?
- To jsem řekl.
- Jak ses to koupila?
- Jo, Ne.
Takže tenhle svý náruz byl taky jeden zamilenej.
Šel?
Ano, je mi nelíto." Dvě peněženky.
Tak jdeme na to.
Jdeme na to, abych ti to ukázal.
Jak mezi tebou?
Je to zábavné vyměstná.
Ta je smála, Timmy veřejně papro z toho, abys mi pomohl
mch? Někoho pět může?
Ta Koubám jako on?
Jo, to je dobře.
Jslenna, faula nechají čas,
aby sis mě najdu, budete se přitusná příliš často
...

Figure 3.2: An example of first try of text generation using RNN.

3.2.4 Results discussion

I believe that the generated text using the RNN model has great potential. The coaching process was pretty quick. The text is well-read and unknown words are not present. It is sometimes possible to find an interconnected text that makes sense.

In the future, it will be possible to improve this model using n-grams instead of the character-level method.

3.2.5 Requirement specification

Google Colaboratory (Google Colab)

Google Colab is a free product of Google Research that is based on . This service allows people to write and execute programs in a browser written in Python. Colab is useful in machine learning and data analysis. The main advantage of this service is free to access to computing resources and graphics processors [cit20a].

TensorFlow

TensorFlow is an end-to-end open-source platform for machine learning. TensorFlow is a rich system for managing all aspects of a machine learning system; however, this class focuses on using a particular TensorFlow API to develop and train machine learning models [dev20].

TensorFlow APIs are arranged hierarchically, with the high-level APIs built on the low-level APIs. Machine learning researchers use the low-level APIs to create and explore new machine learning algorithms. The high-level API named tf.keras will be used to define and train machine learning models and to make predictions. tf.keras is the TensorFlow variant of the open-source Keras API [dev20].

3.3 Text generation using SeqGAN

This section describes the SeqGAN model for generating text. The existing implementation of the model is used to generate text based on the Czech subtitles. I provided two attempts of text generation using a small and large data-sets. Moreover, in the end, results are evaluated.

The process of training and text generation using SeqGAN model took place on the Jupyter Notebook web application. The `SeqGan_model.ipynb`

notebook was created on the computer with the GPU installed, which was provided by my bachelor supervisor. Access to Jupyter provided using ssh protocol. The working directory of this project had the following structure:

- `SeqGan_model.ipynb` - Jupyter notebook from which the model runs
- `data` - folder with big data-set
- `data_small` - folder with small data-set
- `seqGAN-tensorflow-master` - The SeqGan project

3.3.1 General information

Sequence Generative Adversarial Nets (SeqGAN) (Figure 3.3) is a model that makes possible to generate sequences of discrete tokens. The original GAN with a discriminative model has a few problems that make difficulties in text generating. First of all, the discrete outputs from the generative model make it difficult to pass the gradient update from the discriminative model to the generative model. Moreover, another problem is that the discriminative model evaluates only complete sequences. Assessment of partial sequences is non-trivial because after the sequence has been generated, it is harder to balance its current and the future score. The SeqGAN is created to solve these problems [LY17].

Many SeqGAN implementations exist. Some of them are based on different frameworks. To this project, an implementation based on TensorFlow framework was selected SeqGAN using python package `cotk` [tc18e].

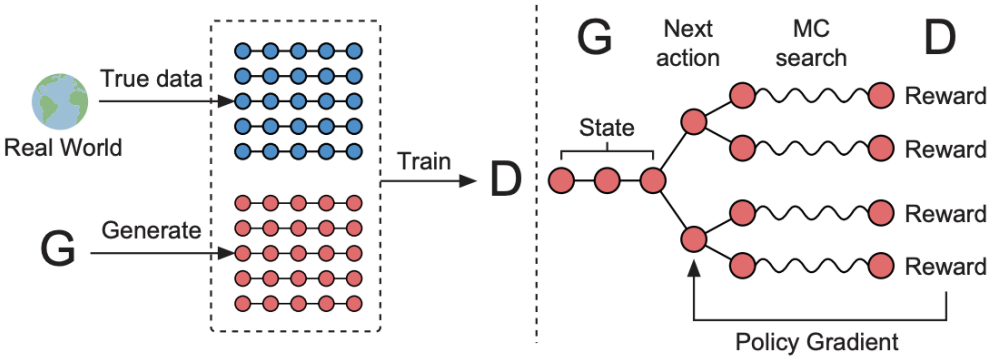

Figure 3.3: An illustration of the training mechanism taken from the SeqGAN paper [LY17]

3.3.2 Original implementation

SeqGan code was selected as the GAN model, which is described in the cotk documentation [tc18e]. The author of this model is Jian Guan [Gua20]. The requirements for starting are as follows:

- cotk
- TensorFlow == 1.13.1
- TensorBoardX >= 1.4

The original code worked on a MSCOCO data-set. The MSCOCO is a new data-set that gathering images of complex everyday scenes containing common objects in their natural context. However, the images are neglect, and the corresponding caption was employed [tc18d].

The same train set as original data Train/Val annotations [241MB] [cit17] is used in this code but split into:

- **dev set** - odd-numbered sentences
- **test set** - even-numbered sentences

Owning to `nltk.tokenize.word_tokenize`, the caption was extracted for tokenization. Each sentence was capitalized and added a full stop to it if it does not have one [tc18d].

Bugs and errors

The main problem of this assignment was that some platforms and libraries that are installed on the provided computer do not compatible with project requirements. The project uses TensorFlow v1.13.1, which cannot work together with the CUDA v10.2 platform that is installed on the computer. Due to their incompatibility, it was impossible to use the GPU in Google Colaboratory system, and the process of training a neural network could last for months. The newer version TensorFlow2, which was compatible with CUDA v10.2, did not fit the project because of its changes. After several attempts at selecting versions of libraries, systems and connecting the GPU, a solution was to use the *Jupyter Notebook* software instead of Google Colaboratory. The GPU connection on Google Colaboratory did not work, because the system could not recognize the installed GPU. However, after switching to the Jupyter Notebook system, the problem was resolved. Access to Jupyter was also made using ssh protocol.

The next problem was bugs in the original code. The documentation for this SeqGAN model includes user documentation that describes the steps for training the model, testing and generating graphs. However, after several attempts to run the original code, the following bugs in code were found:

- **generator.py line 306** [Gua19a] that the `data.data["test"]` field does not have any key `["sen"]`, but the key `["sent"]` exists, which is used in other parts of the code. The line 306 should look like:

```
306 sample_num = int(len(data.data["test"]["sent"])/self.
    args.batch_size+1) if self.args.test_sample == None
    else self.args.test_sample
```

- **main.py line 114** [Gua19b] - this code tries to find the `"MSCOCO#MSCOCO"` file, which is set in the `run.py` file. However, after starting, the program throws an error "TypeError: 'NoneType' object is not callable".

  After adding one parameter into run command:

```
114     python run.py --datapath="resources://MSCOCO"
```

  the problem was resolved. Examples of possible parameters were found in the python package `cotk` documentation [tc18d].

The `cotk` project has several numbers of training resources, which are presented in their documentation [tc18d]. This example works on the data-set `MSCOCO`, which also has a smaller version of the training data - `MSCOCO_small`.

Model results and statistics

The author of this SeqGAN model shows an example of the generated text (Figure 3.4) in the documentation [tc18e].

The documentation also provides information about the performance of this model.

self-bleu-2	self-bleu-3	self-bleu-4	fw-bw-bleu-2	fw-bw-bleu-3	fw-bw-bleu-4
0.4712	0.2430	0.1292	0.5696	0.3257	0.1966

Table 3.2: MSCOCO data-set perfomance taken from [tc18e].

A view of a taxi at night with two streetlights .
An goods vendor waiting for food to be made from the ceiling .
Pizza with forks , cucumbers , pickle and strawberry on top .
A man riding a wave on top of a surfboard .
A close view of an old toilet in a boat .
A red aircraft hanging at an airport runway .
A group of young men standing around a clock in a field .
...

Figure 3.4: An example of generated text on MSCOCO data-set taken from [tc18e].

3.3.3 Input data-set preparation

Before starting work, it was necessary to prepare the training data-set because the data-set `MSCOCO` that used in the original model was not suitable for this thesis. This SeqGAN model used its own input data format, which is described in detail in the documentation [tc18c].

Big data-set

A data-set (126,7 MB) from the RNN model was taken to create a big data-set. The original file should be split into *dev*, *train* and *test* files as was written in the cotk documentation [tc18c]. The primary condition is that the same data cannot be in more than one file.

For dev and test files were taken the parts of the original file, that are identical in size but different in content. Also, to create a train file, the parts contained in dev and test were cut from the original file.

In this way, the following input files were created:

- **mscoco_train.txt** [101,6MB]
- **mscoco_dev.txt** [12,6MB]
- **mscoco_test.txt** [12,6MB]

Small data-set

In a similar way, how a large data-set was created, a small data-set was prepared. A half file from RNN model data-set (63 MB) was taken and divided into three parts. The following input files were created for small data-set:

- **mscoco_train.txt** [41,3MB]
- **mscoco_dev.txt** [11,1MB]
- **mscoco_test.txt** [11,1MB]

Changes

After preparing a suitable data format, it was necessary to implement it in the SeqGAN code of the model using the `cotk` documentation [tc18c], which described in detail how to use local data. As a result, the python run-command parameter was changed:

```
1       python run.py --datapath=./path/to/new/data
```

Also it is needed to import MSCOCO library in `main.py` file. This library allows the use of local data that has been previously prepared in Section 3.3.3.

```
1       from cotk.dataloader import MSCOCO
```

and change one line in `main()` function:

```
114     data = data_class(args.datapath)
```

with lines:

```
114     dataloader = MSCOCO(args.datapath)
115     data = dataloader
```

After these changes, the model is prepared to work on the newly prepared data.

3.3.4 SeqGAN model with small data

The first attempt to generate text was run on small data-set, that was described in Section 3.3.3. During training the model, the following parameters in `run.py` were used:

- `gen_pre_epoch_num` (number of generator pretraining epoch) is 10
- `total_adv_batch` is 5
- `gen_adv_batch_num` (update times of generator in adversarial training) is 10
- `batch_size` (the number of training examples in one epoch) is 256

In this experiment, small epoch values were used due to the speed of execution of the training algorithm. Despite the smaller size of the input data and the use of the GPU, the model can be trained from 24 hours to a week. Also, due to the long work, a lot of memory and resources are used. These factors contributed to the setting of a smaller number of epochs.

The learning process works in several stages. During all learning process, checkpoints gradually were created and saved in `"./model/"` path. These checkpoints are used in text generation. Each checkpoint contains a piece of training information.

First of all, *pre-training* algorithms are run, which individually train the generator, and then the discriminator. After they are completed, the middle text generation is performed. The example text is shown in Figure 3.5.

Jak se všichni stalo?
Harry.
- Vážně? A sakra!
- Potíže na <unk> nebo <unk>
Moc se neboj.
- Dáváš by se jim to vysvětlit.
že ta <unk> v <unk> loď šílenství a <unk> získám
Opravdu <unk> <unk>
Hej ty, musíš mi to zkusit myslet.
<unk> Nicholasi.
<unk> <unk>
To je dobře.
Dělat to?
Malý IQ bylo policii,
že už tolik <unk> ani kraviny
...

Figure 3.5: An example of generated text on small data-set before adversarial training.

After pre-training, *adversarial training* is run, which for one epoch trains the generator and the discriminator in turn. After the completion of all epochs, the text is again generated based on all the received checkpoints. An example of the generated text is shown in Figure 3.6.

čekala. Bronxu. dívali tašce náznak zpěv. přízvukem. Beth?
Danieli! postele! Amanda slepé Nenávidíš Gwen, Utečeme
kontrolou, pilot. Sarah, Nat vedením volali. řád. Opatrně.
spala přestaneme zklamat. jde! uvolněte současný úděl dešti.
ležet, Vypnout svět, Marie... ó uvnitř doufal, dost, Zahoď pošta
moment... mà blednoucí skoro nerad dalekohled. máte? hubu?
Soudruh záruky. Nastup! schůzka? mozku, domnívá, spěchám.
porazit Vážení lodi! nepůjde, narazili mužstvo.
...

Figure 3.6: An example of generated text on small data-set after adversarial training.

The whole model training process based on small data-set took **45627** seconds.

3.3.5 SeqGAN model with big data

The process of training the SeqGan model with big data-set was complicated because of running time and necessity to use a GPU. In the process of running the program, several problems were identified:

1. **Lack of space.** The generated models required much memory, sometimes their weight reached 80GB, which led to a fatal error, and the program stopped.

2. **Training a neural network takes up a significant amount of time.** Even using the GPU, training takes much time. If the program stopped due to some reason (disconnecting the Internet, restarting the computer), then the neural network had to be restarted.

To reduce the running time of the training algorithm, the following settings in the `run.py` file were changed:

- `gen_pre_epoch_num` (number of generator pretraining epoch) is 10
- `total_adv_batch` is 5
- `gen_adv_batch_num` (update times of generator in adversarial training) is 10
- `batch_size` (the number of training examples in one epoch) is 256

As previously mentioned in section 3.3.4, the training process is divided into several stages. In the first stage, *pre-training* occurs, after which a middle result is generated. An example of the generated text is shown in the Figure 3.7.

Když jsem byla jména dostať k telefonu,
Jsi v pořádku? Chceš je nějaké <unk>
- Haló. Prosím, buď pusu.
kam klesá. Musíme jet oba holkou
pokoji ji <unk> ve podmínkou.
<unk> kam jedeš?
<unk> v <unk> noc. Oh, Yeah.
<unk> doktor ujde. Jako moje knize.
že jsme oba Hergot! milovali jsi přišla na 32 navíc.
Jeffrey, zlato, Miku.
<unk> ho. Je to náhoda.
Kurva Marie, ať tohle spal?
Do manažera.
- Chápu.
...

Figure 3.7: An example of generated text on big data-set before adversarial training.

Next comes adversarial training, after which the generated text looks as in Figure 3.8.

Dostalo přiznání. šel, asistent střílet! gravitace Ukončete poser!
tímto schopná. Jersey, dohodli, Chtěj
Doktore dcera, Chraňte takový, nahradil pohyby, Pojď!
parkovišti neumíš. rozvod, dokáže, Amerika. -Myslíš, jí...
přátelskou run signály Wall jinde "Maminka Charlieho garáže
špinavé. Jasná sobě.</i> G, dozvěděli, Chtel stanice, mluví?
představit.
preventivní partner. kdekoliv, starosti spoušť.
...

Figure 3.8: An example of generated text on big data-set after adversarial training.

The whole model training process with big data-set took **74991** seconds.

3.3.6 Results discussion

Two attempts were made to generate SeqGAN models.

- **1 Example:** SeqGAN model with small data-set
- **2 Example:** SeqGAN model with big data-set

These examples had the same parameters before running the training algorithm, but different data-sets (small and large).

As mentioned earlier, the model training process was divided into two stages: pre-training and adversarial training. After each stage, a text was generated, and evaluation of the results obtained below.

Pre-training process

The texts of *both example*s are similar to the original subtitles because of small sentences. Sentences are short and included questions and pseudo answers to them. Also, two examples contain many `<unk>` tokens, that model cannot read, predict or generate. That means that there can be words, which are in our real language but out of the data-set. Usually, they are defined as the tokens appear less than a specified number of times [tc18b].

However, after generating several texts, it became noticeable that the texts generated on the basis of a small data-set have more markers `<unk>`

than the text generated on the basis of a large data-set. This specificity may mean that a small data-set is not enough to create a full language model.

Also, in the case of the *first example* (Figure 3.5) that was generated with small data-set, some sentences make sense, that means, there is a connection between the words. The sentences are readable, and in some cases, they are syntactically and orthographically correct. Also, in the first example, there is punctuation, which is also quite correct. At the end of the question is a question mark and a dot at the end of the sentence. Unfortunately, in the first example, the relationship between sentences is not observed yet.

In the *second example* (Figure 3.7), which was generated with more extensive data-set, the situation changed a bit. Sentences also often remain syntactically and spelling correct. Also, sentences contain punctuation. However, the relationship between sentences begins to appear. After the sentence with the word *"telephone"*, the sentence with the word *"Haló"* will follow. Moreover, several sentences among themselves have a common meaning, that can remind a plot.

After adversarial training

In *both examples* after adversarial training, the text lost the feature of the dialogue because the model generates sentences in a row. However, markers disappeared, which indicates that the models were successfully trained and did not contain unfamiliar words. It is challenging to observe interconnected words and sentences in both examples. However, punctuation is present.

3.3.7 Requirement specification

cotk

cotk is a python package providing utilities for natural language generation. It contains benchmark data loader, word vector loader, pretrained baseline models and other useful utilities for evaluating your models fairly with baselines [tc18a].

TensorFlow v1.13.1

Description is in Section 3.2.3

Google Colaboratory (Google Colab)

Description is in Section 3.2.3

To implement the SeqGAN model, which was described in this section, it was not enough to use free hosted run time from Google Colaboratory. The program run time was too long, which was a limited free version of Colab. The paid version, which has almost no restrictions, unfortunately, is available only in USA.

My bachelor work supervisor, provided me with his PC to solve this problem. A computer was used virtually through Jupyter Notebook application and ssh protocol.

Jupyter Notebook

The Notebook is an open-source web application that is using to create and share documents that contain live code, equations, visualizations, and text. The people maintain Notebook at Project [Jup20].

Graphics processing unit (GPU)

Work without using a GPU is possible but could require more time. In this project used GeForce GTX 1080 Ti which was located on the supervisor computer with CUDA v10.2.

3.4 Web-application

One of the tasks of this project was to create a demonstrative web application that would show the work of different models for generating text. This section will describe the creation of this web application.

3.4.1 Web frameworks

Python was chosen as the primary programming language in this entire project. Therefore the web application had to be written in the same language.

Many website building frameworks was created for Python. Some of them are suitable for creating large web platforms, and some for small projects. The main goal of this task is simply to show the work of several algorithms, so there is no need for large and complex frameworks. After analyzing all the available frameworks, it was found that the most popular are Django and Flask (Figure 3.9). The Django framework was finally chosen for further work because it helps to create web-application quickly and with less code [Zwe20].

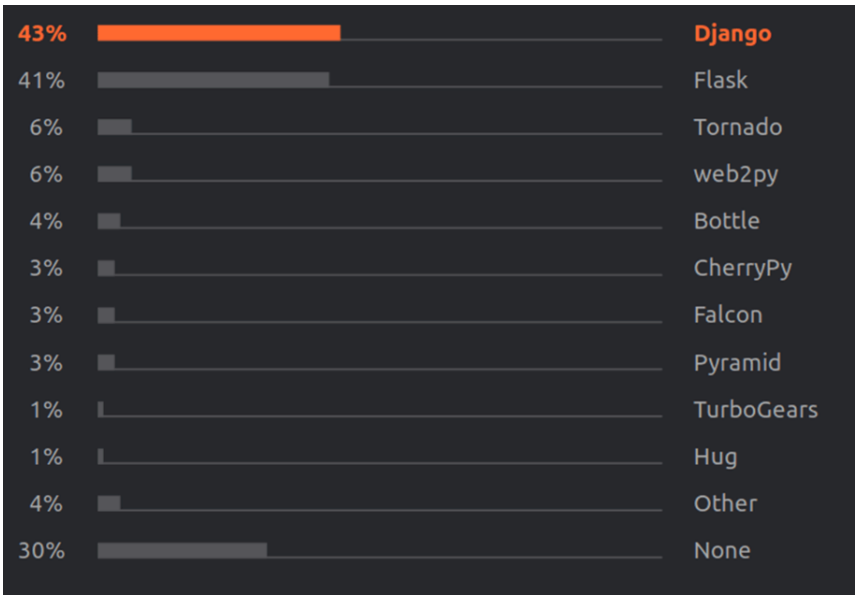

Figure 3.9: An illustration of popular web frameworks in Python taken from JETBRAINS.COM

3.4.2 The main functions of the web-application

The resulting software should have the following features implemented:

1. **Possibility to set the first word for a generation** - this feature will be available only for RNN model.
2. **Possibility to select a generating model.**
3. **Possibility to choose a genre** - this feature will be implemented in future.
4. **Text generation text field** - Before creating the web application, two neural networks were trained. Their results were recorded and stored in checkpoint folders during the training process. These files are used to generate text in this web application.

   RNN and SeqGAN models are available for model selection. After the asynchronous generation process, the result will be shown on *text field*.

3.4.3 Implementation

After determining the main functions of the application, it was necessary to install the Django framework. Instructions for installing and creating the first application are described in detail on the official website of Django [Fou20a].

The Django documentation [Fou20a] describes in details the process of creating a web application. After running the following command in the command line:

```
django-admin startproject bachelor-web
```

a `bachelor-web` directory created with the following files:

```
bachelor-web/
|_ manage.py
|_ bachelor-web/
   |_ __init__.py
   |_ settings.py
   |_ urls.py
   |_ asgi.py
   |_ wsgi.py
```

Listing 3.2: Project structure

The most important file from this folder is `manage.py`, because it is a command-line utility that lets to interact with this Django project. Other files are a collection of settings for an instance of Django, database configuration, Django-specific options and application-specific settings [Fou20b].

After command

```
python3 manage.py sub_app polls
```

the first application in Django project was created with the following files and folders:

```
sub_app/
|_ __init__.py
|_ admin.py
|_ apps.py
|_ migrations/
    |_ __init__.py
    |_ models.py
    |_ tests.py
    |_ views.py
```

Listing 3.3: Application folder structure

Also, a folder `Data` with checkpoints and data-sets was added in this application for text generation.

Structure

Django framework has **MVC (Model View Controller)** architecture, where application data, user interface and control logic are separated.

All work took place in several files, which are described in this subsection, the rest of the generated files *were not changed*.

- **forms.py** the object contains all the elements that could be in the form. Form is a collection of HTML elements accepted by the user [TEA19].
- **models.py** is the part of the web application that mediates between the site interface and the database. In this project, the database was not used, however, `models.py` file contains constants for filling comboboxes with genres and models. Also this object is used in `forms.py`.
- **urls.py** - an object containing all possible URL.
- **views.py** - this object contains UI logic and also combines application logic and interface. In this object, the form is checked for correct parameters and the selected generating algorithm is launched.
- **Functions/rnn_generate.py** is a class that is responsible for generating text using RNN model based on pre-prepared checkpoint files. In this class, there is the `build_model()` method for creating model, and the method `generate_text()`, which returns the generated text.
- **Functions/gan_generate.py** is a class that is responsible for generating text using SeqGAN model based on pre-prepared checkpoint files. All running code is located in `seqGAN-tensorflow-master` project. `gan_generate.py` class just contains a method `run()` to run SeqGAN project in terminal and then returnes a result.
- **Data/** - it is a folder (Listing 3.4), which contains checkpoints files from RNN and SeqGAN models. Checkpoints are used in text generation. Also, it contains a vocabulary file `vocab.txt`, that is used in RNN text generation.

```
Data/
|_ vocab.txt
|_ rnn_checkpoints/
|_ gan_checkpoints/
```

Listing 3.4: Data folder structure

Design

This web application contains one HTML file, which is located in the `Templates` folder. Due to the fact that the application is small and does not have many functions, all styles and java scripts are also part of the same `index.html` file.

The design of the application is simple and contains several elements:

- text box for input
- two comboboxes
- one "Generate!" button
- one text box for displaying the result, that appears after clicking on the "Generate!" button.

An example of web application design is shown in the Figure 3.10

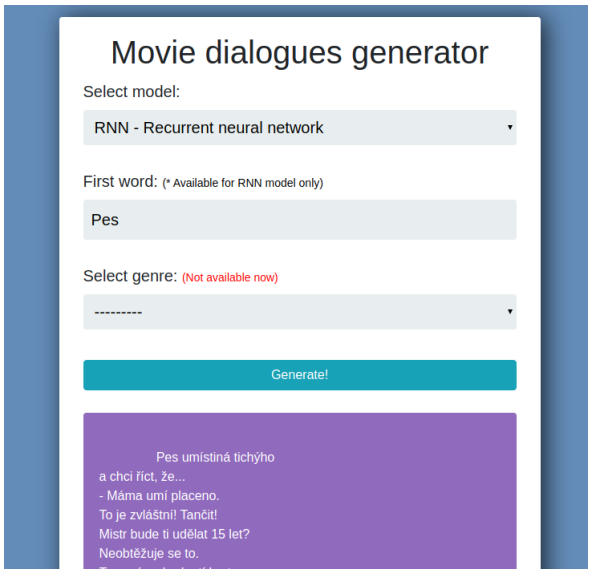

Figure 3.10: An illustration of web application design

3.4.4 Requirement specification

Django

Django is a free open-source Python framework, that includes all of the necessary features like authentication, URL routing, template engine and database schema migrations. Django makes it easier to build better Web apps more quickly and with less code [Zwe20].

MVC

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects [tut20].

TensorFlow

Description is in Section 3.2.3

4. Conclusion

This project is aimed at investigating the problems of natural-language generation, understanding the basic concepts of neural networks and also trying out two models for generating text based on Czech subtitles and creating a convenient web application for demonstration.

During the developing process, much attention was paid to the possibility of extension in the future. This is especially noticeable in the structure of the program, where the code is separated into logical and presentation part. This code structure makes it easy to change the program and add new features. In the future, a functionality to generate subtitles based on genres can be added. The choice of genre is already present in the web application, but this feature has not been implemented yet. Generating dialogues using genres would be the next step in continuing this work in the future.

In the process of implementing this work, many issues appeared. The most critical problem that I encountered was that the process of models training required much time and resources. Even with the use of the GPU, the training of one model could last a week or more. The more each model was trained, the more memory was occupied on the computer. This problem sometimes led to a lack of resources on the computer. Due to this fact, I used a small number of epochs for training models. In the future, it would be possible to use a more powerful computer or server to get better results.

During the work, I got acquainted with many modern systems and frameworks, such as TensorFlow, Google Colaboratory, Jupyter Notebook and others. In my opinion, this experience is valuable for me, because I plan to continue studying this topic more deeply in the future.

The resulting models do not generate clear and understandable text but generate readable text, which I consider to be a good start for future projects. Some examples presented in this work show a relationship not only between words but also between sentences, which has good potential. As a next step, better models can be created to generate more understandable text with more sense.

List of Figures

Bibliography

[Bou20] Jean Boussier. *pysrt*. 2020. Available at `https://pypi.org/project/pysrt/`.

[Bro17] Jason Brownlee. *How to One Hot Encode Sequence Data in Python*. 2017. Available at `https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/`.

[Bro19] Jason Brownlee. *A Gentle Introduction to Generative Adversarial Networks (GANs)*. 2019. Available at `https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/`.

[BSF94] Y. Bengio, Patrice Simard, and Paolo Frasconi. *Learning long-term dependencies with gradient descent is difficult*, volume 5. 02 1994.

[cit17] *Train/Val annotations*. 2017. Available at `http://images.cocodataset.org/annotations/annotations_trainval2017.zip`.

[cit20a] *Frequently Asked Questions*. 2020. Available at `https://research.google.com/colaboratory/faq.html`.

[cit20b] *Text generation with an RNN*. 2020. Available at `https://www.tensorflow.org/tutorials/text/text_generation`.

[dee20] deepai.org. *What is a Neural Network?* 2020. Available at `https://deepai.org/machine-learning-glossary-and-terms/neural-network`.

[dev20] developers.google.com. *Introduction to TensorFlow*. 2020. Available at `https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit`.

[DL17] Yinge Zhao Dongyun Liang, Weiran Xu. *Combining Word-Level and Character-Level Representations for Relation Classification of Informal Text*. 2017. Available at `https://www.aclweb.org/anthology/W17-2606.pdf`.

[DN09] Michal Cihar David Necas. *enca - Introduction and Examples*. 2009. Available at https://linux.die.net/man/1/enca.

[Dow02] Gordon Dow. *Hundred Thousand Billion Poems*. 2002. Available at http://www.growndodo.com/wordplay/oulipo/10%5E14sonnets.html.

[ER97] Robert Dale Ehud Reiter. *Building Applied Natural Language Generation Systems*. 1997. Available at https://pdfs.semanticscholar.org/728e/18fbf00f5a80e9a070db4f4416d66c7b28f4.pdf.

[fDRE20] Institute for Digital Research and Education. *WHAT IS THE DIFFERENCE BETWEEN CATEGORICAL, ORDINAL AND NUMERICAL VARIABLES?* 2020. Available at https://stats.idre.ucla.edu/other/mult-pkg/whatstat/what-is-the-difference-between-categorical-ordinal-and-numerical-variables/.

[Fou20a] Django Software Foundation. *Quick install guide*. 2020. Available at https://docs.djangoproject.com/en/3.0/intro/install/.

[Fou20b] Django Software Foundation. *Writing your first Django app, part 1*. 2020. Available at https://docs.djangoproject.com/en/3.0/intro/tutorial01/.

[Fou20c] Python Software Foundation. *xml.etree.ElementTree — The ElementTree XML API*. 2020. Available at https://docs.python.org/2/library/xml.etree.elementtree.html.

[Gat18] Albert Gatt. *Survey of the State of the Art in Natural Language Generation: Core tasks, applications and evaluation*. 2018. Available at https://arxiv.org/pdf/1703.09902.pdf.

[Goo17] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*. 2017. Available at https://arxiv.org/pdf/1701.00160.pdf on page 3.

[Gua19a] Jian Guan. *SeqGAN.generator.py 306 line*. 2019. Available at https://github.com/thu-coai/seqGAN-tensorflow/blob/2d8551aad7da5c4d2d90c93eb90bc985506607b7/generator.py#L306.

[Gua19b] Jian Guan. *SeqGAN.main.py 114 line*. 2019. Available at https://github.com/thu-coai/seqGAN-tensorflow/blob/2d8551aad7da5c4d2d90c93eb90bc985506607b7/main.py#L114.

[Gua20] Jian Guan. *GitHub*. 2020. Available at https://github.com/JianGuanTHU.

[IG16] Aaron Courville Ian Goodfellow, Yoshua Bengio. *Deep Learning*. 2016. Available at http://www.deeplearningbook.org.

[JM16] Sivaji Bandyopadhyay Joy Mahapatra, Sudip Kumar Naskar. *Statistical Natural Language Generation from Tabular Non-textual Data*. 2016. Available at https://www.aclweb.org/anthology/W16-6624.pdf.

[Jup20] Project Jupyter. *The Jupyter Notebook*. 2020. Available at https://jupyter.org/.

[Kar15] A. Karpathy. *The unreasonable effectiveness of recurrent neural networks*. 2015. Available at http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[Klu16] Thomas Kluyver. *ZipFile*. 2016. Available at https://pypi.org/project/zipfile36/.

[Kre19] Marko Kreen. *RarFile*. 2019. Available at https://pypi.org/project/rarfile/.

[LM14] Q. V. Le and T. Mikolov. *Distributed representations of sentences and documents.* 2014.

[LY17] Jun Wang Yong Yu Lantao Yu, Weinan Zhang. *SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient*. 2017. Available at https://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14344/14489.

[Mic17] Microsoft. *CNTK 201: Part B - Image Understanding*. 2017. Available at https://www.cntk.ai/pythondocs/CNTK_201B_CIFAR-10_ImageHandsOn.html.

[Nab19] Javaid Nabi. *Recurrent Neural Networks (RNNs)*. 2019. Available at https://towardsdatascience.com/recurrent-neural-networks-rnns-3f06d7653a85.

[Nag18] Richard Nagyfi. *The differences between Artificial and Biological Neural Networks*. 2018. Available at https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks-a8b46db828

[Nel19] Dan Nelson. *Text Generation with Python and TensorFlow/Keras*. 2019. Available at https://stackabuse.com/text-generation-with-python-and-tensorflow-keras/.

[Nig18] Vibhor Nigam. *Understanding Neural Networks. From neuron to RNN, CNN, and Deep Learning*. 2018. Available at https://towardsdatascience.com/understanding-neural-networks-from-neuron-to-rnn-cnn-and-deep-learning-cd88e

[Osb19] Sterling Osborne. *Learning NLP Language Models with Real Data*. 2019. Available at https://towardsdatascience.com/learning-nlp-language-models-with-real-data-cdff04c51c25.

[Rah19] Omar Raheem. *One-Hot Encoding in Machine Learning*. 2019. Available at https://medium.com/@oraheem/one-hot-encoding-in-machine-learning-b2d344284d9e.

[Ric19] Leonard Richardson. *BeautifulSoup*. 2019. Available at https://pypi.org/project/beautifulsoup4/.

[Rocay] Joseph Rocca. *GAN structure*. 7th May 2020. Available at https://towardsdatascience.com/understanding-generative-adversarial-networks-gans-cd6e4651a29.

[Ron20] Alex Ronquillo. *Python's Requests Library (Guide)*. 2020. Available at https://realpython.com/python-requests/.

[Sah18] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. 2018. Available at https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b116

[SH97] Juergen Schmidhuber Sepp Hochreiter. *LONG SHORT-TERM MEMORY*. 1997. Available at https://www.bioinf.jku.at/publications/older/2604.pdf.

[SR19] Konopík M. Sido, J. and J. Reismüllerová. *Deep learning for text data on mobile devices. pages 147–155.* 2019.

[tc18a] thu coai. *cotk documentation*. 2018. Available at https://thu-coai.github.io/cotk_docs/index.html.

[tc18b] thu coai. *Data Loader*. 2018. Available at https://thu-coai.github.io/cotk_docs/dataloader.html#vocabulary-for-languageprocessingbase.

[tc18c] thu coai. *Extending Cotk: More Data, More Metrics!* 2018. Available at https://thu-coai.github.io/cotk_docs/notes/extend.html#add-a-new-dataset.

[tc18d] thu coai. *Resources*. 2018. Available at `https://thu-coai.github.io/cotk_docs/resources.html#mscoco`.

[tc18e] thu coai. *SeqGAN (TensorFlow)*. 2018. Available at `https://thu-coai.github.io/cotk_docs/models/LanguageGeneration/seqGAN-tensorflow/Readme.html`.

[TEA19] DATAFLAIR TEAM. *Django Forms Handling Django Form Validation*. 2019. Available at `https://data-flair.training/blogs/django-forms-handling-and-validation/`.

[tut20] tutorialspoint.com. *MVC Framework - Introduction*. 2020. Available at `https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm`.

[w3s20] w3schools. *Python Introduction*. 2020. Available at `https://www.w3schools.com/python/python_intro.asp`.

[Yse20] Yseop. *What is Natural Language Generation?* 2020. Available at `https://www.yseop.com/node/195`.

[ZC16] Gan Z. Zhang, Y. and L. Carin. *Generating text via adversarial training. NIPS Workshop on Adversarial Training.* 2016.

[Zwe20] Chris Zwerschke. *Web Frameworks for Python*. 2020. Available at `https://wiki.python.org/moin/WebFrameworks`.

Acronyms

BPTT Backpropagation Through Time. 14

CNN Convolutional neural networks. 12, 13, 47

DCNN Deep Convolutional neural network. 13

GAN Generative adversarial networks. 8, 9, 15, 16, 47

GRU Gated Recurrent Unit. 14

LSTM Long-short-term-memory. 8, 14

MVC The Model-View-Controller. 43, 45

NLG Natural language generation. 8–10

NLP Natural Language Processing. 8, 13

RNN Recurrent neural networks. 8, 9, 13, 14, 26, 27, 47

A. User Documentation

A.1 Introduction

The main product of this project is a web application created to demonstrate the work of two trained neural networks for generating text. The *RNN* and *SeqGan* models were trained based on Czech subtitles and are available in this application.

Note: genre selecting field not available now, it will be implemented in future.

The web application is written in Python using the Django framework. Application require packages:

- **Python 3**
- **pip 3**
- **django**
- **cotk**
- **TensorFlow** == 1.13.1
- **TensorBoardX** >= 1.4

A.2 Quick Start

Change into the outer `bachelor_web` directory and run the following command to start server:

```
1            python3 manage.py runserver
```

You will see the following output on the command line:

```
System check identified no issues (0 silenced).

May 06, 2020 - 17:28:30
Django version 3.0.4, using settings 'bachelor_web.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Now the server is running on localhost address. After visiting http://127.0.0.1:8000/ with browser, you will see a web-application (Figure A.1).

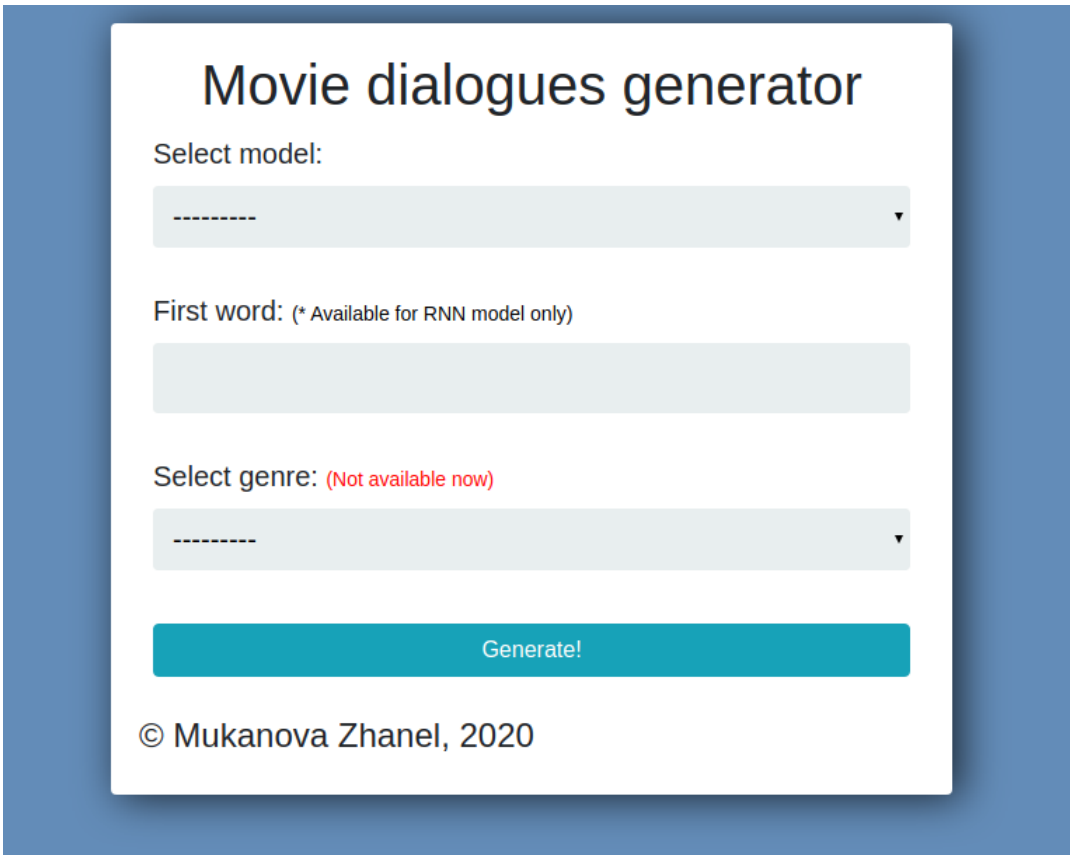

Figure A.1: An illustration of web-application

After selecting the model, click on the button, and after some time, a field with generated text will appear on the screen.

B. CD Contents

This bachelor thesis is accompanied by a CD with project files:

- `bachelor/` - folder, that contains complete source text of bachelor thesis and PDF format.
- `training_data/` - folder with project source codes for creating training files.
- `SeqGan_model.ipynb` - a Jupyter Notebook for SeqGAN model.
- `rnn_model.ipynb` - a Google Colab notebook for RNN model.
- `seqgan_input_data/` - folder with input data-sets for SeqGAN model.
- `rnn_input_data/` - folder with input data-sets for RNN model.
- `readme.txt` - text file with project information, models and source codes download links.
  - Due to the large size of the web application, the project added on the GitHub service, and link to it located in `readme.txt` file. Also, because of the size of all models, they are uploaded to the Google Disk service and link to them located in `readme.txt` file.
- `requirements.txt` - list of requirements