Use of evolution of deep neural network for text summarization

Author

Antoni Pieta

Advisor

Josep Carmona

Department of Computer Science

Co-advisor

Włodzimierz Funika

AGH University of Science and Technology

MASTER IN INNOVATION AND RESEARCH
IN INFORMATICS - Advanced Computing
FACULTAT D'INFORMÀTICA DE BARCELONA (FIB)
UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC) – BarcelonaTech

Date of defense
1 July 2021

Abstract

In the era of internet, the ability to quickly extract useful information out of big amounts of data has become an important capability. This includes text summarization, a Natural Language Processing task of compressing a given text into a shorter one in such a way that it is consistent with the original text, concise, correct and as informative as possible. The leading solutions of this problem use various Deep Neural Networks. Designing an optimal DNN's architecture is a difficult task requiring a lot of expertise, time and work. In this work I attempt to facilitate this process using coevolution of neural networks. I use Pytorch-dnnevo framework to find networks capable of solving NLP tasks, including text summarization using coevolution. I implement architectures based on RNN, LSTM and Seq2seq with attention mechanism. Metrics like ROUGE-N, BLEU and F1 as well as datasets like IMDb Movie Reviews and Amazon Fine Food Reviews are used. The results show that, given suitable layer types, coevolution is capable of constructing networks that can solve NLP tasks. It can help engineers find the optimal architecture and hyperparameters for a given dataset.

Contents

1 Introduction **3**
- 1.1 Background . . . 3
- 1.2 Motivation . . . 4
- 1.3 Research objectives . . . 4
- 1.4 Feasibility study . . . 5
- 1.5 Structure overview . . . 6

2 State of the art **7**
- 2.1 Text summarization . . . 7
  - 2.1.1 Problem definition . . . 7
  - 2.1.2 Example . . . 7
  - 2.1.3 Specific tasks . . . 8
- 2.2 Approaches . . . 8
  - 2.2.1 Extractive . . . 9
  - 2.2.2 Abstractive . . . 10
- 2.3 Datasets . . . 11
- 2.4 Metrics . . . 11
- 2.5 Summarizing with neural networks . . . 12
- 2.6 Recent papers . . . 14

3 Methodology **17**
- 3.1 Evolution of neural networks . . . 17
  - 3.1.1 Neural Architecture Search . . . 17
  - 3.1.2 Evolutionary algorithms . . . 18
  - 3.1.3 Evolution of neural networks . . . 19
  - 3.1.4 Coevolution of neural networks . . . 19
- 3.2 Overview of stages . . . 20
- 3.3 Pytorch-dnnevo framework . . . 21
  - 3.3.1 Front-end . . . 21
  - 3.3.2 Server . . . 23
  - 3.3.3 Workers . . . 24
- 3.4 Layers . . . 24
- 3.5 Data preprocessing . . . 27

4 Development and results evaluation **28**
4.1 Conducted experiments . . . 28
4.2 Computing environment . . . 29
4.3 Basic evolution experiments . . . 30
4.4 Sentiment analysis experiments . . . 33
4.4.1 Sentiment analysis using default layers . . . 33
4.4.2 Sentiment analysis using RNN . . . 34
4.5 Text summarization experiments . . . 36
4.5.1 Qualitative analysis . . . 36
4.5.2 Quantitive analysis . . . 40

5 Conclusions **44**
5.1 Research objectives . . . 44
5.2 Limitations . . . 45
5.3 Future research . . . 45

Bibliography **47**

Chapter 1

Introduction

This chapter provides a brief overview of the domain of the problem. I explain why research in this topic is needed and what is my motivation. Also, I propose the initial research goals that I will pursue.

1.1 Background

In the past decade, Artificial Intelligence (AI) has developed drastically in numerous fields. This was caused mainly by the progress in the area of Deep Neural Networks (DNN). It was driven by numerous factors described in Minar and Naher [2018]. New implementations like Krizhevsky et al. [2012] allowed using Graphics Processing Unit (GPU) clusters in order to train deep networks over big datasets, increasing the accuracy greatly. Breakthroughs in the research on this topic, like dropout (Srivastava et al. [2014]), new network architectures and activation functions were an important factor as well. Neural networks were able to displace some existing traditional algorithms by providing better solutions or even tackle tasks that were too difficult beforehand (Sarker [2021]).

This change was so important and influential that it has not only been noticed by computer scientists using these new possibilities, but also by users of electronic devices and the internet. Nowadays, users take advantage of these solutions on a daily basis. Computer Vision (CV) is used in order to unlock phones (Li et al. [2020]) and recognize objects in the surrounding to provide data for autonomous cars (Janai et al. [2017]). Voice recognition enables users to invoke commands without making physical or eye contact with their devices. Solutions relying on Big Data are used to suggest the most suitable content in social media and the most relevant products in online stores (Lv et al. [2017]).

One of the main areas that are developed recently is Natural Language Processing (NLP). The most common tasks are Machine Translation (MT), participating in a conversation as a chatbot, performing searches, topic classification, sentiment analysis, checking grammatical and spelling correctness, information extraction and text summarization (Otter et al. [2019]). Networks like Recurrent Neural Networks (RNN) that are able of taking the position of a token in a sequence into account are well suited for these tasks (M.Tarwani and Edem [2017]). Even more interesting results are provided by Long Short-Term Memory (LSTM) networks which are able to perceive relationships between words that are far away from each other in

long texts (Young et al. [2018]). Transformer can process multiple tokens from the sequence at the same time which speeds up training (Vaswani et al. [2017]).

Designing neural networks is, however, a difficult process. It often requires testing many architectures and comparing their results. In order to evaluate each of them, it first has to be trained on a large dataset, which requires a lot of computations. Searching for the most suitable architecture is frequently done by manually configuring and running these trainings. In order to this facilitate this process, various techniques have been developed, including reinforcement learning (Baker et al. [2017]) and the evolution of neural networks (Stanley [2017]) that will be used in this study.

1.2 Motivation

In the era of internet, accessing the data is not difficult anymore. Instead, the ability to extract useful information out of it is has become a bigger challenge. When it comes to texts, readers often decide that obtaining full information on a topic is not necessary, or at least not worth spending the time required to read the whole content. Instead, they prefer to get the main point of the text in much shorter time (Moran [2020]). In order to address that, many newspapers decided to include a short paragraph or a couple of bullet points at the top of their articles that sum up the content best. Authors of comments on the forums often tend to include a "Too Long - Didn't Read" (TL;DR) section in their comment (Volske et al. [2017]). Reports are created of some books, especially the ones that students have to read compulsorily at school.

Nevertheless, such summaries are provided by authors of texts only on some occasions. Otherwise, readers have to extract the information themselves, which requires time and effort. They often attempt to achieve that by reading just a couple of sentences or extracting some keywords. These summaries cannot be perfect, because in this case the summarizer does not have the full knowledge of the text, which is necessary to extract the most important information. Therefore, having the ability to generate reliable summaries automatically would be an important achievement.

While neural networks are able to solve many tasks on a level that is already fairly satisfactory, text summarization still poses a challenge. Even the best solutions that are currently available are not consistent in constructing meaningful and accurate summaries. It is understandable, as such network not only needs to be able to understand the meaning of the input, but also to generate a natural language text as the output. It is difficult to include all the important information without facing self-repetition or inventing false information (Verma and Verma [2020]). They also have problems with grammatical correctness. Therefore, further research in this field is needed.

1.3 Research objectives

Even though text summarization is only one of many NLP tasks, it is a popular, broad area. Solutions are currently intensively developed by teams in the biggest research centers using models trained on enormous text corpora. Therefore, taking into account my resources and

knowledge, it is understandable that achieving novel results and establishing new state of the art scores in these tasks would be close to impossible.

Nevertheless, my research resides on the intersection of two areas: NLP and coevolution of neural networks. This gives me a possibility of establishing goals and posing questions that are much more specific than simply trying to achieve the highest benchmark score. Moreover, I decided to adopt multiple goals, so that they can be achieved to various extent, depending on the circumstances.

Firstly, I would like to assess if networks for NLP tasks can be developed with coevolution. Then, more specifically, assess if text summarization models can be designed in this way. Depending on the answers to these questions I will either explore various ways in which such evolution can be performed or identify the limitations that impede this process from succeeding. I also want to verify if using coevolution reduces the amount of expertise that an engineer needs to have in order to develop useful neural networks.

Besides these scientific goals, I also decided to set some personal ones. I would argue that they are even more important for me as this study is part of my education path. Firstly, I would like to learn as much as possible about the field of neural networks and NLP. I also hope to get familiar with new tools, frameworks and programming techniques that I have had no experience beforehand and improve my skills in these that I already know. Besides these technical competences I would also like to get more experienced in project management and gain soft skills as this is by far the biggest project that I have ever attempted to complete.

1.4 Feasibility study

As mentioned above, this is my first project of such a scale. Moreover, it is an individual work, contrary to previous projects that I participated in during my studies. Due to that, it is important for me to evaluate the risks that can prevent me from completing the study.

The first category of possible risks is related to my competences prior to the beginning of this project. While I have some knowledge about Machine Learning that I gained during my classes, I have little actual experience in conducting experiments in this field. As a result, I might make some wrong decisions about the direction of my research or assign too little time for some phases of the work. Moreover, I will use a number of tools and frameworks that I will need to learn while working on the main topic, which might further slow down my workflow.

The second type of problems might arise due to the fact that my work will be heavily based on an evolution framework that is being written by my co-supervisor as a part of his doctorate. As the project is still in development it is reasonable to assume that it might contain some errors. Also, it has no documentation which means that I will have to get all my knowledge about it from the co-supervisor or by reading the code itself.

The last category of risks might be related to the fact that neural networks usually need a lot of computing power. Performing the calculations on my personal computer might be impossible in a reasonable time. Therefore, I will possibly need to use a computational cluster which creates some potential problems, related to getting access to the cluster, learning how to use it and most importantly designing a way in which the framework can be executed using it.

1.5 Structure overview

This document is divided into five chapters. The second one provides an overview of current state of knowledge in the field of text summarization. The third introduces the concepts and tools that I use and describes my approach to the problem. The fourth focuses on conducted experiments, listing their configurations and analysing the results. Finally, the fifth concludes the findings of the work.

Chapter 2

State of the art

In this chapter I describe in detail the field of text summarization. I introduce the core ideas, give some examples, explain the most important solutions that are used extensively and review some recent works.

2.1 Text summarization

2.1.1 Problem definition

The goal of text summarization task is to take a text as an input and generate a shorter text, in such a way that the meaning of the output is as close as possible to the meaning of the original, and that it conveys as much important information as possible. It can be viewed as a lossy compression problem, which differs it from Machine Translation. Depending on the application, there can be different expectations regarding the length of the generated text. The most common source texts used in the domain are news, articles, internet posts, opinions and comments.

2.1.2 Example

In order to better understand text summarization, we will analyze the following example taken from Amazon Fine Foods Reviews dataset:

Text 2.1: *Oh, this is some good cat food! Have you ever opened a can of Friskies and gagged at the awful smell? How can your cat eat that? This cat food smells pretty good, and it simply looks and smells like good food. My kittens made so much noise when I opened that can! They gobbled it right up, and I have to admit that it smelled pretty good.*

This can be a valuable review, where the author expresses a number of things. By analysing how much does the author focus on each of them we can estimate their importance in the following order:

1. general positive impression,
2. good smell,

3. cats' appetite,
4. negative opinion about other product - Friskies,
5. good appearance.

Even though the content of the review might be valuable, the delivery could be less elaborate. The word "good" appears four times, including three expressions about smell. The information about Friskies and cats' appetite were a bit longer than necessary. If a user is trying to evaluate a product quickly, reading multiple reviews, it would be easier to read a summary instead. Depending on the desired length, it would probably include some of the listed information, expressing them concisely:

Text 2.2: *This food smells and looks good, my cats love it! Much better than Friskies.*

Or, even shorter:

Text 2.3: *Good smelling cat food!*

This work will focus on generating such summaries automatically with the use of coevolution of neural networks.

2.1.3 Specific tasks

Before explaining how such summarization can be done automatically, it is worth mentioning that while the definition above describes text summarization in general, there are a couple of specific tasks distinguished in this field:

- Extreme Summarization, which is the process of fitting the information into one-sentence output text. It can be used for headline generation.
- Multi-document summarization, which allows extracting knowledge from a number of texts on one topic. This method is particularly interesting as it goes beyond simple shortening of a text; it includes gathering knowledge from various sources and allows the reader to get familiar with a new topic quickly.
- Multi-modal summarization, which merges the areas of NLP with other branches of data processing, like Image Recognition. These models are able to combine the knowledge extracted from a text with information conveyed by other means in order to create a more precise summary. This can easily find a practical application as most articles and papers include pictures or videos to help the reader understand the content better.

2.2 Approaches

There are two main approaches to automatic text summarization: extractive and abstractive.

2.2.1 Extractive

In case of extractive summarization, the goal is to select the sentences that are most important and informative without rephrasing anything. This approach can be also described as a task of sentence classification: for each sentence from the input, the model has to decide if it should be included in the summary or not. This method was invented by Luhn [1958].

Below are some possible summaries of text 2.1 obtained this way. I intentionally list both good and bad examples to show some properties of this approach. Note how they are composed of full sentences from the original text:

Text 2.4: *Oh, this is some good cat food!*

Text 2.5: *Oh, this is some good cat food! This cat food smells pretty good, and it simply looks and smells like good food. They gobbled it right up, and I have to admit that it smelled pretty good.*

Text 2.6: *Oh, this is some good cat food! How can your cat eat that? My kittens made so much noise when I opened that can!*

- Advantages:
  - Grammatical correctness is guaranteed, as long as the original text was correct. Examples: texts 2.4, 2.5, 2.6.
  - The models can be very simple. The most naïve solution is extracting every n-th sentence. Example for n=2: text 2.6.
  - It is usually fast.
- Disadvantages and challenges:
  - Satisfactory summarization can be impossible to achieve for some source texts. This is especially evident for short, for example one-sentence summaries. It might be impossible to find a sentence in the source text that would briefly convey all the important information.
  - There is no possibility of removing non-essential information or stylistic phrases from the chosen sentences. Examples: "Oh" in text 2.4, "I have to admit that" in text 2.5.
  - Despite its simplicity, it still does not guarantee to prevent repetitiveness. Example: text 2.5 - smell is mentioned multiple times.
  - It does not guarantee to prevent logical incoherences. In natural language the sentences often contain references to each other, for example pronouns refer to objects introduced earlier. After removing part of sentences, these relations do not make sense anymore. Examples: text 2.5 - "They" refers to "kittens" that are not mentioned in this summary; text 2.6 - "that" refers to "Friskies".

2.2.2 Abstractive

On the contrary, summarising abstractly consists of analysing the source text and generating the result from scratch, i.e. without explicitly copying parts of input directly to output. This means that the result might contain sentences or even words that are not present in the original source. Texts 2.2 and 2.3 represent this approach, as well as the following ones:

Text 2.7: *This cat food smells and tastes good! It smells good. Good smell.*

Text 2.8: *This cat food smells and looking good!*

- Advantages:
  - Any summary can be obtained, including the best possible, because they are constructed without any assumptions regarding included words and their order. Example: text 2.2 is arguably a good one, because it conveys all important information in two short sentences.
  - Words or phrases that do not appear in the source text can be used. Examples: "love it" and "better than" in text 2.2.
  - It is possible to construct summaries of various lengths, including shorter than one sentence. This allows Extreme Summarization (explained below). Example: text 2.3.
- Disadvantages and challenges:
  - This approach is much more difficult - both conceptually and computationally.
  - Models often hallucinate potentially erroneous information that was not present in the source text. In practical uses this is a particularly dangerous flaw as it might mislead the reader. Example: information about taste in text 2.7.
  - Grammatical correctness is a challenge. Example: "looking" instead of "looks" in text 2.8.
  - The models tend to repeat the same information multiple times in the output. Example: text 2.7.
  - Out Of Vocabulary (OOV) words need to be processed correctly. Example: "Friskies" in text 2.1.
  - The common solution which is sequential processing is difficult to parallelize and therefore takes a lot of time for longer documents, regardless of the number of processing units.

The disadvantages of the extractive approach prove that it is fairly limited. It is difficult to make further improvements in these models. Because of that, most current research focuses on the abstractive one. Even though there are many challenges that come with it, it is possible to mitigate them with proper techniques.

2.3 Datasets

Developing models for summarization (especially training those based on neural networks) requires using large datasets of ground truth data, i.e. text-summary pairs. Below are a few of them that are commonly used in this research field. The sources are given in parentheses:

- Reddit TL;DR - Reddit publications and comments with summarization provided by the authors themselves. (Volske et al. [2017])
- CNN/DailyMail - press news summarized in a few sentences. The most commonly used dataset in past few years. (Hermann et al. [2015])
- Amazon Fine Foods Reviews - reviews of products in an online store. (McAuley and Leskovec [2013])
- Newsroom - news dataset with 1.3 million articles. (Grusky et al. [2018])
- LCSTS - Chinese news dataset. (Hu et al. [2015])
- MLSUM - news with summaries in 5 languages that can be used for multilingual summarization. (Scialom et al. [2020])
- XSum - dataset for extreme summarization. (Narayan et al. [2018])

2.4 Metrics

Apart from models and datasets it is also necessary to have tools to measure their performance. They can be used both for comparing one method to another and for training the model. Over the years many metrics have been developed:

- ROUGE-N (Recall-Oriented Understudy for Gisting Evaluation) introduced in Lin [2004] that calculates what part of n-grams from reference summary are also represented in candidate summary (n=1 or n=2 is often used):

  $$\text{ROUGE-N} = \frac{\sum_{gram_n \in Ref} Count_{Cnd}(gram_n)}{\sum_{gram_n \in Ref} Count_{Ref}(gram_n)} \tag{2.1}$$

  where Ref and Cnd are the reference and candidate summaries and $Count_{Sum}(gram_n)$ is the number of n-grams $gram_n$ in summary Sum.

- BLEU (BiLingual Evaluation Understudy), originally used in Machine Translation, is similar to ROUGE, but focuses on precision instead of recall. It calculates what part of n-grams in candidate text are also present in reference text. Lin and Hovy [2003] applied this metric to summarization in the following way:

  $$C_n = \frac{\sum_{gram_n \in Cnd} Count_{Ref}(gram_n)}{\sum_{gram_n \in Cnd} Count_{Cnd}(gram_n)} \tag{2.2}$$

$$Ngram(i,j) = BB * exp(\sum_{n=i}^{j} w_n log C_n) \quad (2.3)$$

where $4 \geq j \geq i \geq 1$, $w_n = \frac{1}{(j-i+1)}$ and BB is a brevity bonus, set to 1. For $Ngram(k,k)$, the metric calculates precision for k-grams. For $Ngram(1,4)$, the metric calculates a weighted score for n-grams of length 1 through 4.

- F-Score - a combination of precision and recall:

$$\text{F-Score} = \frac{(\beta^2 + 1) * P * R}{\beta^2 * P + R} \quad (2.4)$$

where P is a precision metric like C_n (2.2) and R is a recall metric like ROUGE-N (2.1). β regulates the balance between recall and precision and is set to 1 for a common F_1 score. (Nancy Chinchor)

- ROUGE-L (Lin [2004]) is calculated using the Longest Common Subsequence (LCS) of summaries, i.e. the longest sequence of words that appears in both summaries:

$$R_{lcs} = \frac{LCS(X,Y)}{len(X)} \quad (2.5)$$

$$P_{lcs} = \frac{LCS(X,Y)}{len(Y)} \quad (2.6)$$

$$\text{ROUGE-L} = \frac{(\beta^2 + 1) * P_{lcs} * R_{lcs}}{\beta^2 * P_{lcs} + R_{lcs}} \quad (2.7)$$

where $LCS(X,Y)$ is the length of the Longest Common Subsequence of summaries (X,Y) and $len(X)$ is length of summary X.

- Human evaluation - the most accurate and the most expensive method. It is rather a method of evaluating the results that can be formulated as a metric rather than a metric itself.

2.5 Summarizing with neural networks

There are some NLU tasks like sentiment analysis or classification which can be approached with typical tools known from other fields like dense layers, convolutional layers or recurrent neural networks. The basic solution is to use them to map the input to the output that has the size of the number of classes. This approach does not apply to tasks like Machine Translation or text summarization that require not only the ability to understand the input, but also to generate the output, i.e. Natural Language Generation (NLG).

Sutskever et al. [2014] proposed a method to achieve that which was later extensively used in other works. The authors call this method sequence-to-sequence or Seq2seq. Its schema is shown in the figure 2.1. They use two RNN layers. The first one act as an encoder which sequentially processes the input creating a hidden state vector. After reaching the end of the sequence, this hidden state is passed to the second RNN, which acts as a decoder. In each

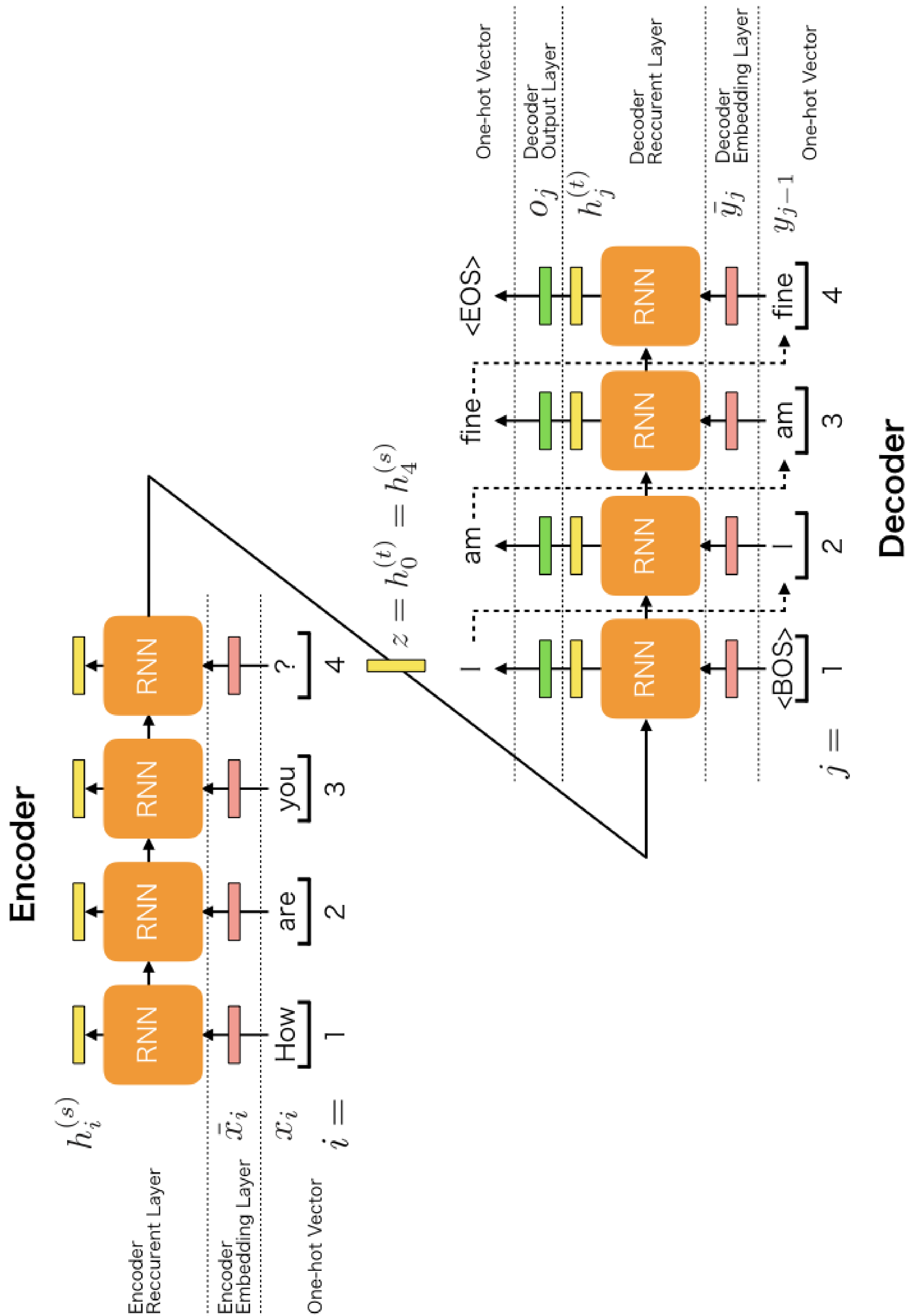

Figure 2.1: *A schema of Seq2seq model (source: Chainer framework documentation, https://docs.chainer.org/en/stable/examples/seq2seq.html)*

time step it produces one output token basing on hidden state and the previously generated token. In case of training, it is possible to feed the reference summary tokens instead.

The Seq2seq model itself has a major flaw: it tends to forget the general meaning when the sequences are long. While working on Machine Translation in Bahdanau et al. [2014], the researchers noticed that the model gives better results when it is able to focus more on certain words and less on the others. This is intuitive, because words in the input language have their equivalents in the other language, often in an one-to-one relationship. They achieved this functionality by introducing an attention mechanism to the decoder which "aligns" input and output sequence so that it performs the translation in a more local manner.

Later, this mechanism was successfully introduced to text summarization in Rush et al. [2015] and Nallapati et al. [2016]. Even though by definition not every word has its own corresponding word in the summary, it is possible to map only part of the original words to summary words, or map multiple original words to one word in the output.

Recently, Language Models (LM) gained popularity in the field of NLP. They are networks trained on large datasets containing just natural language text. LMs can be used in various tasks by treating them as pre-trained networks and fine-tuning them afterwards so that they fit the tasks better. Such a model, called Transformer, was proposed in Vaswani et al. [2017].

The research described above formed a backbone of how summarizing with neural networks is performed. Since then, many other works have been published that base on them, but by applying certain modifications or ideas they attempt to obtain better results or focus on a certain problem. While I will most probably rely on the core of text summarization knowledge in my work, I decided to learn about more recent work as well and describe them in the following section.

2.6 Recent papers

These works often intend to address some of the challenges of abstractive summarization described in the section 2.2. One of them is how to handle Out Of Vocabulary (OOV) words, i.e. words that appear in source that are missing from the embedding and the model does not know their meaning, like names or domain-specific terms. In other tasks like sentiment analysis sometimes they can be simply omitted with the hope of extracting sufficient information from the rest of the text. In case of text summarization they are often essential to let the reader understand the main entity or topic of the original text.

In order to cope with that, a copy mechanism was introduced to Seq2seq in Gu et al. [2016]. In each step, the model named CopyNet calculates the probability that determines if it should generate a word or copy it from the source. This allows for OOV tokens to be present in the generated text even though the model does not really "understand" them. It can be said that this solution is a mix of abstractive and extractive approach.

Another challenge of text generation is the tendency to repeat information in the output. In Tu et al. [2016] authors decided to tackle it using a coverage mechanism. They define a simple measure of coverage loss. This parameter measures how much attention did the decoder give to words from the source document in previous steps. It reduces the probability that the network will decide to focus on the same information multiple times.

The copying and coverage mechanisms were both applied with modifications by See et al.

[2017]. They propose a pointer-generator network that behaves partially like a regular abstractive generator and partially like a pointer network. Similarly to the copy mechanism, a part of the words are copied to the output text. The generation is done with a sequence-to-sequence LSTM encoder-decoder that calculates a probability distribution over a dictionary and then chooses the most probable word. This model is likely to copy the important, specific information and at the same time it is capable of paraphrasing more general information (the meaning of the text), connecting or splitting phrases into grammatically correct sentences. Authors also argue, that this solution reduces the risk of hallucinating information.

A work that is particularly interesting for me is Liu et al. [2017], because it also makes use of an idea of two agents competing against each other. Authors propose a generative adversarial network composed of two neural networks: generator and discriminator that play a minimax game trying to win with each other. The task of the generator in this case is to take an input text and generate a summary that is as close to human-written summaries as possible. The discriminator's aim is to classify an input summary as written by human or machine. Both networks are trained during the process. As they become better in completing their tasks, they pose a more difficult challenge to the opponent, forcing it to learn as well. At the end of the learning phase it is the generator that is responsible for performing the summarization. Authors argue that their solution addresses three problems present in abstractive text summarization: trivial and generic summaries, limited grammaticality and readability and error accumulation over time present in the common method of next word prediction (caused by the fact that the model bases on words that it generated earlier). The model performance was tested on CNN/DailyMail obtaining mostly better results than other methods (Nallapati et al. [2016], See et al. [2017], abstractive deep reinforced model) in ROUGE 1, 2 and L metrics as well as human comparison.

In Chen and Bansal [2018] authors decided combine extractive and abstractive approaches to accumulate their advantages. The summarization is done in two steps by two models trained separately. Firstly, the most salient sentences are extracted using a LSTM-RNN. As the datasets usually contain only the source text and a human-written abstract summary, it was necessary to find a way to construct a reward function to be able to train the network. For each sentence in the target summary, a sentence with the highest ROUGE-L score from source text is selected. In other words, the extractor tries to select these sentences from the source that are the most similar to the corresponding sentences in the target summary. Then, each of the chosen sentences is processed by an encoder-aligner-decoder abstractor that behaves similarly to the one used in a regular abstractive approach. The difference is that in this step the model does not reduce the number of sentences; it paraphrases them in an one-to-one manner.

Khandelwal et al. [2019] attempts to use a modification of the Transformer model proposed by Vaswani et al. [2017] as a pre-trained network for the text summarization task. The authors modify it and join the encoder and decoder together, in such a way that the input is composed of both source text and the target separated by a delimiter. They achieve three goals: (1) ensuring all model weights, including those controlling attention over source states, are pre-trained, (2) avoiding the redundancy of loading copies of the same pre-trained weights into the encoder and decoder, (3) using fewer parameters compared to encoder-decoder networks. Combining these ideas (pre-training with LM and joining encoder with decoder) make this model perform better than both not-pre-trained models and various configurations of

pre-trained encoder-decoder models that did not join the two parts. An important advantage of this model is not only how it performs at the end, but more importantly that it is sample efficient. Even after learning using only 1% of the dataset (3000 samples) it already achieves competitive scores. This may be important for application in domains where the training data is not easily available.

Usually the models are trained on datasets with human-written summaries and therefore are trained to mimic such examples. The problem is that even if the network is trained perfectly, the obtained summary can still not be optimal. This is because the summaries from internet datasets like Reddit TL;DR can be of low quality. In order to solve this issue, in Stiennon et al. [2020] researchers decided to use human feedback to rate summaries generated by the model in addition to using summaries from the dataset. The four steps that form the process are: (1) training an initial summarization model using human-written summaries dataset, starting from GPT-style transformer models trained on text from the internet, (2) assembling a dataset of human comparisons between summaries (i.e. knowledge about which summaries are perceived as better), (3) training a reward model, whose aim is to rate if a given summary is likely to be appreciated by humans, (4) fine-tuning the summarization models with RL to get a high reward from the model from the 3rd point. This gave much better results not only than simple supervised learning, but also human-written reference summaries. Nevertheless, the drawback of this solution is that comparing the summaries requires expensive human labour, as mentioned in section 2.4.Related work has been done by Bohm et al. [2019]

Finally, in Maynez et al. [2020] authors decided to gather results from the most common summarization models (pointer-generator, models based on transformer and BERT) and analyse what are the most frequent issues. The task was extreme summarization - the output texts have to contain just one sentence. Instead of only relying on popular simple metrics like ROUGE, they performed a human evaluation of faithfulness and factuality. They also applied more advanced metrics like textual entailment and question answering to check if they will be able to reflect the factuality better than other metrics. The analysis showed that 70% of summaries contain hallucinations. Most of them are extrinsic, which means that the hallucinated content cannot be inferred from the source text. Most of these are erroneous. These numbers show that paying more attention and verifying the factuality is necessary and should be developed in future work. Pre-trained language models yield results that are much more factual. It also is shown that commonly used metrics like ROUGE often score high even if the text contains a lot of hallucinations. Authors point out that a metric of textual entailment performs much better. Surprisingly, the question answering metric does not behave well in this case.

Chapter 3

Methodology

In this chapter I describe my approach to the problem. I explain the mechanism of coevolution of neural networks and introduce the package Pytorch-dnnevo that I will be using to perform the experiments. I describe what experiments I am planning to conduct and what I intend to achieve with each of them. I define layers and tools that I will need to add to the framework and I present my configuration for performing the computations.

3.1 Evolution of neural networks

3.1.1 Neural Architecture Search

Creating a neural network capable of solving a given problem essentially consists of two stages. Firstly, the engineer should choose the right network architecture that will have the potential of storing the knowledge needed for the given type of data. Then, this knowledge needs to be encoded into the network, by finding optimal weights for the connections present in the model. With the increasing difficulty of tasks that neural networks are designed to tackle, both of these stages become more challenging as well.

The latter stage usually requires processing big amounts of training examples through the network and then applying a backpropagation algorithm that modifies the weights. The engineer needs to provide enough computational power and a sufficiently big dataset. This process has its challenges, but good algorithms are well-known and widely present in various machine learning frameworks. This means that a regular user can train a chosen network not knowing how it is exactly done on a low level.

Meanwhile, the process of designing new architectures of networks and choosing the optimal hyperparameters is a difficult task that is usually done manually. It requires professional experience and expertise. An engineer with little experience in machine learning will probably not be able to diagnose why their network performs poorly and how it can be enhanced. Sometimes, they might try to experiment with different architectures by using various layer types, orders, activation functions and connections. While it might improve the performance, it is a time-consuming work.

This challenge sparked interest in the field of Neural Architecture Search (NAS). It is a process of automated design of a network for a certain problem. It makes use of three components:

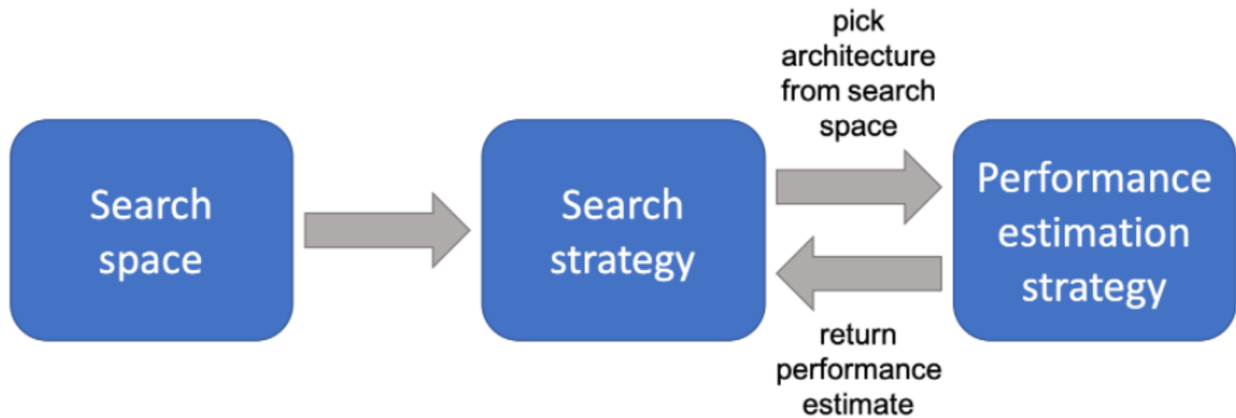

Figure 3.1: *The dependency schema between three components of Neural Architecture Search*

- Search space, where the set of neural networks that is taken into account are defined. Properties like the maximum size of network or a set of eligible types of layers can be defined.
- Search strategy, which defines the approach used to explore the search space.
- Performance estimation strategy, which permits evaluating the performance of neural networks chosen from search space by the search strategy. In some cases, the goal is to be able to evaluate the model without actually training it.

The relationships between these three components are shown on the figure 3.1.

The description above is a general overview of how architecture search is performed. In practice it can be implemented in many ways. Some of the known approaches are: reinforcement learning, supernetwork training and the one that will be used in this work - evolutionary algorithms.

3.1.2 Evolutionary algorithms

Evolutionary algorithms (EA) are the algorithms that rely on mechanisms similar to the ones that form biological evolution, including mutations, selection and reproduction. These algorithms apply well to problems that can be solved with a heuristic solution. The results are rarely expected to be optimal.

The possible solutions to a problem form a population, where the individuals differ from each other. The initial population does not consist of all of the possible individuals. Instead, is is a sample that can be initialized randomly or manually by defining some known solutions that are prone to improvements. Then, there is a fitness function capable of predicting if an individual is a potentially good solution.

The evolution process consists of many iterations. In each of them, the individuals are reproduced using crossover in order to obtain new potential solutions. Mutations can also appear in some of them. Then, they are evaluated by the fitness function. The ones that

scored the lowest are eliminated from the population. The rest is used to perform the next iteration. These iterations are performed until the best-fit individuals perform in a satisfactory way.

3.1.3 Evolution of neural networks

As mentioned above, Evolutionary Algorithms can be used in Neural Architecture Search. In this case, the possible solutions that form the population are various architectures from the search space. The performance estimation strategy serves as the fitness function in evolution. The mechanism of manipulating the population is a search strategy.

In practice, the crossovers can for example be done by concatenating parts from two candidate neural networks. The mutations can be implemented in a number of ways:

- adding or removing a layer,
- adding or removing a connection,
- changing the type of a layer,
- changing the size of a layer,
- changing other hyperparameters of a layer, for example convolution's kernel size or the number of channels,
- changing the hyperparameters of training, like the learning rate.

Having clarified how is the population formed and manipulated during the evolution, the remaining question is how to design a fitness function, or a performance estimation strategy. The most basic solution is training the individuals over the training set and returning the score achieved on a validation set. This way, we can be sure that the scores assigned to the individuals are as close as possible to the score that will be finally achieved on the test set.

3.1.4 Coevolution of neural networks

Even though using the fitness function described above is perfect in terms of relevance and accuracy of the fitness function, it has an important disadvantage. In order to leverage the power of evolution, the population should consist of many individuals that are evaluated over many iterations. Training each one of them over full dataset is often not possible due to the time required to perform one training. This is why often a more efficient fitness function needs to be designed.

Obtaining the score faster would be possible by training the individuals over a small part of the dataset. However, this could make the model be biased towards the contents of this subdatasets. Many valuable, difficult examples (e.g. residing close to the border between two classes) would be omitted. To address that, coevolution mechanism can be used instead of a basic evolution.

Coevolution algorithm is a more complex version of evolution. Instead of enhancing one group of individuals of a certain type, two different populations evolve at the same time. Their

tasks, and what follows their fitness functions, oppose each other causing the populations to compete with each other. It is similarly based on mechanisms that can be found in nature.

In case of NAS, the first population is the same as in regular evolution - a set of candidate networks. It is called the main population. The second group is composed of various subsets from the training dataset, where each subset is one individual. These individuals are called Fitness Predictors (FP), as they are used to predict the fitness of the networks from the main population.

Each iteration consists of two stages. The first one is similar to an iteration in classic evolution - the networks from the main population are reproduced, evaluated and selected into a new version of the population that will be used in the next iteration. However, to make the process faster, the fitness of each network is obtained not on the whole training set, but only on the most difficult subset known so far, i.e. the best individual from the Fitness Predictors population. How the most difficult subset is selected is explained in the next paragraph.

In the second stage, evolution of FP population is executed. Firstly, the individuals are bred. Then, the best network from the main population is evaluated over all the Fitness Predictors that form the second population. In this case, the obtained scores are not used to evaluate the network, but the Fitness Predictors. Moreover, the goal of a Fitness Predictor is to obtain the lowest score possible. Then, the individuals that scored the lowest are bred into the new version of FP population. The objective here is to select and breed the subdatasets that are the most difficult for the network to process correctly.

As a result, both populations improve their performance over time. The Fitness Predictors try to "beat" the networks by trying to select the most difficult examples from the dataset; the networks try to "beat" the Fitness Predictors by correctly solving the training examples that form them. This way the training process is fast and the training examples are challenging and do not exclude important parts of the training set.

The individual that received the highest score in the last iteration is the final network chosen in the evolution process. To check its true abilities it should be trained one more time on the whole dataset to eliminate any bias that could have appeared due to training on a small subset.

3.2 Overview of stages

In my work I will perform a Neural Architecture Search of networks that are capable of solving NLP tasks, and more specifically, text summarization. I will use coevolution of neural networks as a realization of NAS. In order to do that, I will need to design and implement all three components of NAS.

The first component, search space, requires defining a set of possible individuals that will form both populations. In case of main population, this will include implementing various layers like Dense, CNN, RNN, LSTM, Seq2seq that can be combined to form a network architecture. Apart from simply defining them, there are some restrictions that I will need to ensure. Some layers cannot be connected to each other, and even if they can, sometimes the tensor needs to be reshaped between them. In case of FP population, I will download and preprocess the datasets so that they can be sampled and fed to a network during training.

For both populations I will also need to restrict the sizes of individuals.

The implementation of search strategy, i.e. manipulating the populations in my case will be provided by Pytorch-dnnevo framework described in section 3.3. Because of that, this part will most probably not require a lot implementation changes. I should be able to influence this part by tweaking the parameters.

The last part, performance estimation strategy or fitness function, require defining how the networks from the main population are evaluated using a given FP (and vice versa). I will need to implement the training and evaluation process that will work for all possible networks. I will do that by defining an according forward function for all the defined layers. Then, for each of the defined datasets, I will implement a loss function capable of evaluating how well does the output of the model match the dataset's reference label.

After designing such functionalities I will perform experiments and analyse their results to answer questions of this work. First, I will tackle the problem of sentiment analysis on IMDb Movie Reviews dataset. Then, I will approach text summarization on Amazon Fine Foods Reviews dataset. By tweaking evolution's parameters I want to influence the exact behaviour of each NAS component.

3.3 Pytorch-dnnevo framework

In my work, I am using a framework called Pytorch-dnnevo (https://gitlab.com/pkoperek/pytorch-dnn-evolution). This is a tool written by my co-supervisor which is capable of performing the coevolution. The framework itself consists of three parts described in the following sections:

- front-end,
- server,
- workers.

Apart from that, it also uses a number of other components:

- Postgres database,
- Celery distributed system,
- RabbitMQ message broker,
- Flower monitoring tool.

3.3.1 Front-end

The first part is an interface displayed on a page that can be opened in a browser. It serves as point of interaction between the user and the rest of the framework. It is based on React.JS.

It provides two main functionalities. The first one is shown on figure 3.2. The user can schedule an evolution experiment by providing its configuration. This can be done by filling the fields or by providing a json object. The experiment configuration contains parameters

Figure 3.2: *Control page of Pytorch-dnnevo interface where an evolution experiment can be scheduled*

that define the behaviour of the evolution. They are divided in two parts, one per each population. Each part has the following fields (default values in parentheses):

- Individual size (FP: 2000, Main: 16),
- Population size (FP: 16, Main: 32),
- Crossover probability (0.5),
- Mutation probability (0.02),
- Training configuration:
  - Batch size (128),
  - Dataset (MNIST),
  - Learning rate (0.1),
  - Momentum (0.0),
  - Optimizer (sgd),
  - Dense layer size multiplier (1),
  - Convolution layer size multiplier (1),
  - Number of iterations (10).

Apart from these, experiment name and maximum number of evolution iterations have to be specified.

The second functionality is displaying the results of the experiment. They are presented on a graph, where for each iteration the average, minimum and maximum of the scores of individuals from both populations are displayed as well as the standard deviation. They are updated in real time as the calculations are being performed. This gives the user a possibility of evaluating quickly if the experiment is executing correctly and if it has the potential of bringing interesting results.

3.3.2 Server

The core of the evolution process is managed by the server - a Flask application. It is responsible for storing information about populations, managing evolution iterations, scheduling the evaluation of the individuals via the broker, interpreting the results, modifying the individuals and communicating with front-end.

When it receives a command from the front-end, it starts a new evolution experiment. First, it creates the initial populations of networks and Fitness Predictors. The networks are represented in the form of genotypes. The Fitness Predictors are simply lists of indices that point to specific examples in the training dataset.

Then it starts performing evolution iterations. For each one of them, the following steps are executed:

1. Perform crossover between pairs of individuals from the main population with a probability specified in the experiment configuration.

2. Perform mutation on the individuals from the main population with a probability specified in the experiment configuration.

3. Schedule the evaluation of the individuals from the main population against the best of the Fitness Predictors unless they have already been evaluated in the previous steps. These tasks are scheduled using a RabbitMQ message broker.

4. Collect the results from workers.

5. Select the individuals that achieved the highest scores to form the main population for the next iteration.

6. Send the scores to the front-end.

7. Save the results in the database.

8. Execute steps 1-7 once again, but this time for the Fitness Predictors population (in the third step, the Fitness Predictors are evaluated against the best of the networks from the main population).

3.3.3 Workers

The work that requires the most computational power, which is evaluating the individuals is not done in the server part. These calculations are performed by workers that run independently. Thanks to that, it is possible to run multiple workers at the same time so that they evaluate multiple individuals in parallel. These workers are managed by Celery system that can be monitored using an interface provided by Flower.

Whenever the server wants to schedule an evaluation, it sends a message to the broker. Each message contains a network genotype, training dataset's subset (a Fitness Predictor) and the training configuration. Then, it distributes the messages to running workers. Once a worker receives it, it performs the following steps:

1. Translate the genotype into a network.
2. Load the training examples specified by the Fitness Predictor.
3. Perform training of the network over these examples.
4. Return the result to the server.

The genotype is a list of genes. Each gene represents one input or output of a layer and is given by five numbers in the range of [0, 1):

- X, Y - the coordinates that are used to determine which layers should be connected to each other,
- Gene type - it determines if this gene represents an input or an output of a layer.
- Extra - a value that is interpreted individually depending on the layer's type. By default, in case of Dense layer it influences the output size, and in case of Convolutional layer, the number of channels.
- Layer type - it determines the type of the layer.

There are two layer types available by default in the framework: Dense and Convolutional. There are also two image recognition datasets: MNIST and CIFAR10. This means that I will have to implement any other layers and datasets that I need to use. Also, the process expects the dataset to be provided in a specific format with sizes of inputs and outputs explicitly defined, which will possibly need to be changed for NLP datasets.

3.4 Layers

In my work I will be using various types of layers. The most basic one is a dense, or fully-connected layer, which is one of the most commonly used in ML. Each output neuron is connected to all input neurons. The weights of each connection determine how much does the output neuron depend on linked input.

Convolutional layer is most commonly used in image recognition, but can also be applied in other tasks. It focuses on small areas of input called kernels or filters and attempts to

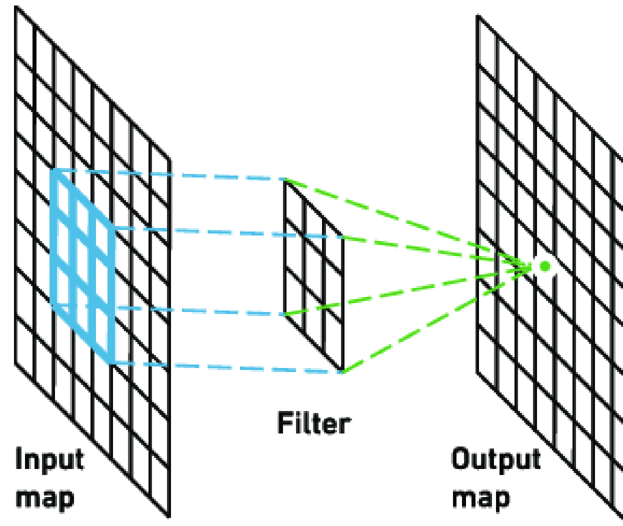

Figure 3.3: *The architecture of a Convolutional layer with a single channel. A filter of size 3x3 detects features in input map and stores this information in output map. (source: Yakura et al. [2017])*

detect features that are present in them. This creates a map of local features that serves as the output. Multiple maps, called channels, can be created this way, each describing a different feature in the input. Its architecture is shown in figure 3.3.

In case of Pytorch-dnnevo framework both one- and two-dimensional convolutions are implemented. The latter is applied only if the input of the layer is known to be two-dimensional, for example coming from other two-dimensional convolutional layer. Otherwise, one-dimensional convolution is applied.

A commonly used network in sequence processing is RNN. It processes the input sequence token by token. In each step, it combines the input token with its hidden state. This state serves as a memory capable of remembering the information. The network keeps this state across timesteps, or in other words, passes this state to itself to be used in the next timestep. The architecture is shown in figure 3.4. Even though it is called a network, I will use it and refer to it as a layer. because due to its nature it would not be possible to construct it from layers in an evolution experiment.

RNN has a major disadvantage - it tends to forget information in long sequences in a process called vanishing gradient problem. To eliminate it, an enhanced version was proposed, called Long Short-Term Memory (LSTM). While the core idea is the same as in RNN, an additional element called cell is added. It is passed along the timesteps just like the hidden state, but is managed in much more precise manner by gates. They decide what information should be forgotten, added and sent to output from the cell. An architecture is shown in figure 3.5.

In order to tackle summarization I needed to add a layer that would be able to learn sequence generation instead of classification. I decided to use a Seq2seq model (Sutskever et al. [2014]) with attention (Bahdanau et al. [2014]). It is described in section 2.5. In this case I also decided to use this model as a single layer.

These are the basic versions of layers that I will be using. They can, however, be modified

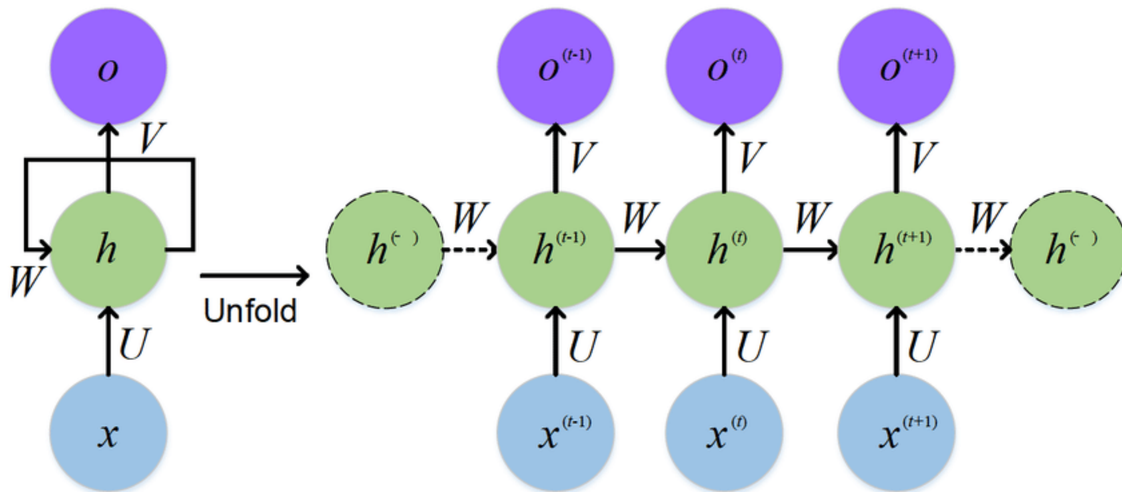

Figure 3.4: *The architecture of Recurrent Neural Network (RNN). It is presented in two manners: on the left the concise, time-independent representation. On the right, it is unfolded along the time axis. X represents the input sequence, h the hidden state and o the outputs at each timestep. (source: Feng et al. [2017])*

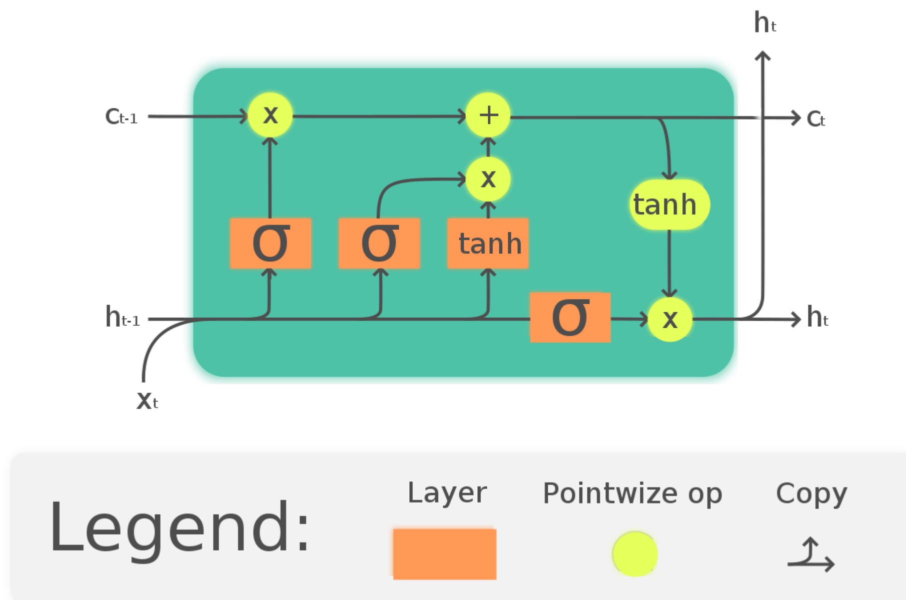

Figure 3.5: *The architecture of LSTM. The cell, represented by the horizontal "c" line, is modified only by two gates. X represents the input sequence and h the hidden state. (source: Guillaume Chevalier - LARNN: Linear Attention Recurrent Neural Network, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=99599411)*

in numerous ways. RNNs (and LSTMs) can contain more that one hidden layer. They can also be designed to process tokens in reverse order apart from the normal flow. To prevent overfitting, dropout of various rate can be applied to Dense and Recurrent layers. Overfitting occurs when the model fits the training data too closely and remembers too specific details instead of generalizing the knowledge. Dropout can help with this problem by excluding some random group of neurons for each example during training.

3.5 Data preprocessing

It is not possible to directly feed textual data into a neural network. The network input is an array of real values of fixed length, meanwhile the data is in the form of strings. Therefore, the text needs to be tokenized to break up the string into a list of words, optionally cleared from stopwords, converted to a list of embeddings and padded into fixed-size array. This preprocessing is applied to both source text and the summaries.

Tokenization is a process of converting the input string into a list of tokens. The naive way of performing it is simply splitting the input string on spaces. However, while tokens are similar to words, they can also be other symbols like numbers or punctuation. Therefore, it is better to use a tokenizer capable of handling various cases and exceptions of the natural language, like SpaCy tokenizers.

Stop words are the most common words of a language that do not have much lexical meaning; instead, they serve as function words that help establish grammatical relationship between other words in a sentence. These words are often removed during preprocessing to reduce the number of computations. In case of text summarization the removal is optional - the effect that it has on model's scores have to be observed to determine if it is better to perform it or not. It is important to only remove them from source texts and not from target summaries as it would impede the model from constructing summaries that contain stop words.

Embedding is a representation of a token in a form of a real-valued vector. The words with similar meaning should be represented by vectors that are close to each other in the vector space. The dimension of the vector is an arbitrarily chosen number, typically between 50 and 300. While it is possible to start with randomly distributed embedding values for all tokens, it is more efficient to use embeddings pre-trained on large corpora like GloVe (Global Vectors for Word Representation). In this case, only OOV tokens need to be initialized randomly.

Padding is a process of unifying the examples' lengths so that they can be organized into arrays of fixed size. To achieve that, special PAD tokens are appended to the end of examples that are shorter than the limit. The examples that are too long are trimmed to the maximum size.

Chapter 4

Development and results evaluation

4.1 Conducted experiments

Abstractive text summarization is a difficult task. It requires the model to both understand the source text and compose a new one. Because of that, I decided to progress through a couple of intermediate stages. I hoped that this will give me a better understanding of the evolution process, implementation details of the framework and the tools used in NLP.

Firstly, in order to get familiar with the framework I performed some experiments using the functionalities that were implemented by default, without changing the code responsible for the evolution process and network evaluation. This included running some experiments on MNIST dataset. MNIST is a labeled set of images of hand-written digits that is commonly used as a typical Machine Learning task. The input images are of size 28x28 and the output vector is a one-hot vector of size 10, one number per each digit.

Then, I wanted to get acquainted with the issue of processing sequences. To do that, I decided to run evolutions for an NLP task less difficult than text summarization. Apart from introducing me to the subject, I wanted to determine if and how NLP tasks in general can be solved using coevolution, which is one of the research goals mentioned in the section 1.3.

As such task, I chose sentiment analysis, which does not require constructing output text. It has the goal of determining if the author of a given text has a positive or negative opinion on a topic. I used the IMDb Movie Reviews dataset which contains 50.000 textual reviews of movies, each paired with a one-hot vector of length of two which indicates the opinion. I performed the experiments gradually to observe how the accuracy depends on the amount of effort put into adapting the framework to the task. I started by applying minimal changes to the framework that were necessary for it to be able to to process the dataset. Then, I introduced some NLP-specific layers like RNN, LSTM and other enhancements.

Finally, I approached text summarization. I used the Amazon Fine Foods Reviews dataset. It contains over 550.000 reviews of products paired with short summaries. In this case it was more difficult to take a gradual approach similar to the sentiment analysis. It was not possible to start with simple layers, because they are not capable of generating the desired type of output - a sequence. Therefore I decided to implement a full Seq2seq model and allow the evolution to tweak it. I also needed to change both the loss function and the dataset loading mechanism so that they were capable of operating on output sequences instead of category labels.

4.2 Computing environment

As said before, coevolution of neural networks is able to reduce the amount of training computations compared to regular evolution by evaluating the networks on small datasets. However, it should be noted that it still needs a lot of computational power as the task of Neural Architecture Search in general requires more computations than simply training a single network.

PLGrid is a Polish nationwide infrastructure that enables scientists carrying out complex calculations. At the time of his experiments, my co-supervisor used a PLGrid Cloud 2.0 environment which gives the user access to Virtual Machines with root access. This was well-suited for the framework, which is a complex system that requires installing a number of tools (npm, Postgres, Celery, RabbitMQ), contrary to the regular Machine Learning work which often only requires Python with some domain packages like Tensorflow.

Unfortunately, this service is being withdrawn and I did not have the possibility of using it. I could only use the main PLGrid computing cluster, which allows user to schedule jobs on CPU and GPU nodes. Having learned that the scheduling is done via a strict queuing system (SLURM) and that there is no root access provided, I assumed that running the framework there would be impossible or very difficult. Because of that, I tried to use my personal computer.

At first, even though it was a slow and flawed process, I managed to conduct some experiments, for example with simple default evolutions on MNIST dataset. Unfortunately, as I tried to head towards more demanding tasks like sentiment analysis or text summarization, it became clear that it will not be possible to continue using a notebook.

I investigated the possibilities of conducting my research using commercial solutions. I had previous experience with Google Colab which lets the users freely perform computations on CPU, GPU and even Tensor Processing Units (TPU). Nevertheless, I quickly discarded the idea of using it because it is designed to execute Jupyter notebooks and would not be suitable for setting up a complicated environment.

Then, I tried to evaluate cloud services like Amazon Web Services (AWS) or Google Cloud Platform (GCP). They both enable users to create powerful Virtual Machines in cloud. Unfortunately as I wanted to execute some experiments I realized that my resource needs not only exceeded the limits provided in as a trial, but also considerably exceeded funds that I was willing to spend.

After consulting my case with PLGrid Helpdesk I learned that there is a Singularity service available on the computing cluster. It allows user to run programs in containers, including Docker images, even if they are not installed in the system. I considered a couple of potential solutions using it, which included setting up the workers in the cluster and connecting to them from the server on my personal computer. My co-supervisor objected to that, because the framework's server is not prepared for the case of losing connection with workers. I decided to run all components in the cluster. It took a lot of time to configure all the components, adapt the framework and learn how to run them as SLURM jobs. Finally, I managed to reach a setup that allowed me to run workers on multiple nodes which was necessary to get the desired speedup. I wrote two scripts that are meant to be executed as two SLURM jobs:

- The first one is responsible for setting up and running Postgres database and RabbitMQ broker in Singularity as well as starting the Pytorch-dnnevo server. It also exports the name of the node that it is executing on to a file.

- The second script reads the node name from this file and then starts a worker that listens to the broker. This script should be scheduled using the "-N" option, which runs the same code on many CPU nodes, causing multiple workers to listen to the same broker.

Both scripts redirect outputs of each component to separate files which allows the results to be checked even after closing and reopening the terminal. Apart from these scripts I wrote a third one, which evaluates a given individual over the whole dataset. It should be executed to obtain the real score of the individual chosen by the evolution (or any other individual if needed).

The cluster can be accessed via Secure Shell Protocol (SSH) connection in a terminal, which means that the front-end of the framework cannot be viewed directly. I decided not to use the interface and read the results of the experiments by browsing the database, where all scores from all iterations are stored.

Using this configuration I managed to execute the jobs using Dense and Convolutional layers with a proper speedup as all the networks from one evolution iteration evaluated in parallel on different nodes.

Nevertheless, I discovered that when using RNNs the learning was still too slow. Training a single network, even on a small subset, took a couple of hours. I noticed that training them on GPU nodes was much faster, allowing me to run evolutions with Recurent Networks. In order to obtain a sufficient amount of GPU computing hours I needed to apply for a grant in the PLGrid infrastructure. After I received it, I set up the framework again, but on the cluster with GPU available.

4.3 Basic evolution experiments

At first, in order to check if the code is running properly, I used very small populations and individuals (rest of parameters set to default):

- FP population size: 4
- FP individual size: 4
- FP population training iterations: 1
- Main population size: 4
- Main individual size: 4
- Main population training iterations: 1

The accuracy oscillated around 10% over all iterations, which is the expected score for guessing randomly over 10 classes. This was in line with expectations, as the networks barely got

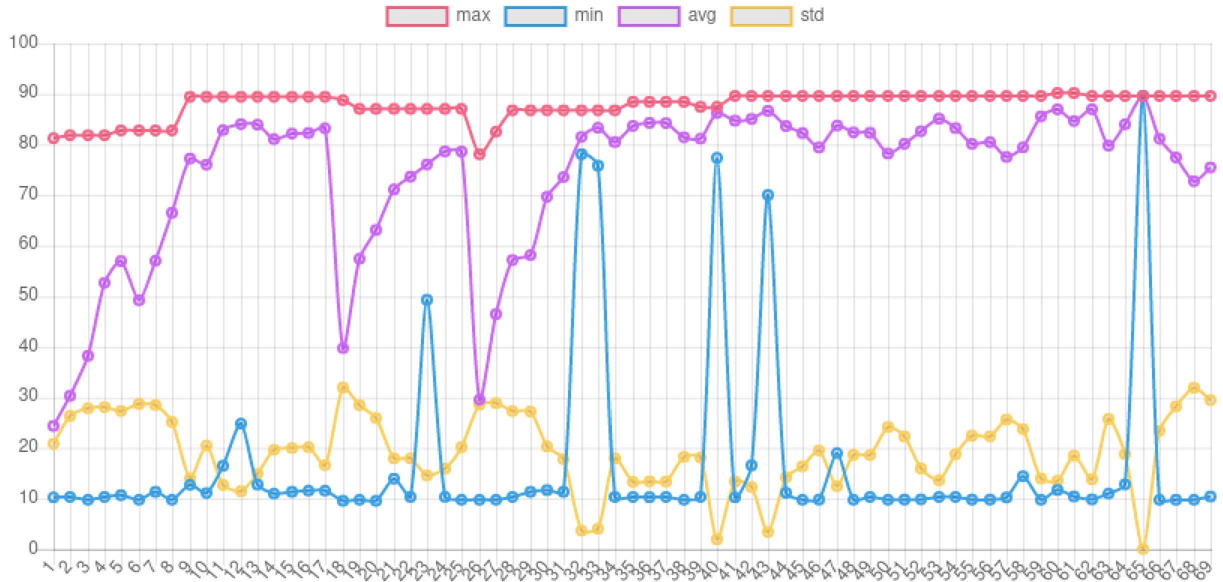

Figure 4.1: *Main population scores in the evolution experiment over MNIST dataset. This experiment was conducted on my personal computer, so I could view the graph in the Pytorch-dnnevo interface.*

any training.

Then I increased the parameters to the default values for this dataset to train bigger networks over bigger subsets:

- FP population size: 2000
- FP individual size: 16
- FP population training iterations: 10
- Main population size: 32
- Main individual size: 16
- Main population training iterations: 10

I conducted a number of experiments with similar configurations. The results of one of them are shown on the figure 4.1. As expected, the networks were able to learn to distinguish the digits with much higher accuracy. The best individuals achieved a score of 80% already in the first iteration. This fact alone, however, did not prove that the evolution had brought effects - it happened mainly due to the fact that each individual was trained over a bigger subset and over more iterations.

The most important conclusion was that the accuracy increased over time. The average accuracy improved from 25% to over 80% and the maximum accuracy from 80% to 90%. This proves that over time evolution was able to construct networks that had more potential solely due to their architecture.

No.	Type	Input size/channels	Output size/channels	Dependencies
0	Conv2D	28x28, 1 chn	26x26, 1 chn	-
1	Dense	676	2	0
2	Dense	678	1	0, 1
3	Conv1D	679, 1 chn	677, 1 chn	0, 1, 2
4	Dense	677	10	3

Table 4.1: *Architecture of a network evolved to recognize MNIST dataset examples with 89.75% accuracy using default parameters*

No.	Type	Input size/channels	Output size/channels	Dependencies
0	Conv2D	28x28, 1 chn	26x26, 12 chn	-
1	Conv2D	26x26, 12 chn	24x24, 1 chn	0
2	Dense	576	10	1

Table 4.2: *Architecture of a network evolved to recognize MNIST dataset examples with 89.55% accuracy using modified parameters*

The architecture of one of the networks that resulted from these experiments which scored 89.75% is shown in table 4.1. The values in "Dependencies" column indicate the layers that have their output linked to the given layer. For example, here the outputs from layers 0, 1 and 2 are concatenated to feed the layer 3.

Knowing that it is possible to achieve higher results on MNIST dataset I tried to change some parameters. As seen in the architecture in the table 4.1, the sizes of outputs of Dense layers and the number of Convolution channels are all small. This is because initially I was using the default values of 1 for "Dense size multiplier" parameter which influences the output size of the Dense layer and for "Convolution size multiplier" parameter which influences the number of output channels of the Convolution layer. Therefore I decided to try bigger values for these parameters, hoping that the network will be capable of storing more information. At first, this caused the output sizes to accumulate to huge values in the subsequent layers. Because of that, I reduced the size of networks. I also increased the number of training iterations as the loss did not seem to be stable before the end of each training.

- Dense size multiplier: 100
- Convolution size multiplier: 10
- Main individual size: 6
- Training iterations: 20

While these changes did not improve the accuracy results, the architecture of networks changed in a desired way. As seen in table 4.2 the network is shorter and more channels are used in the Convolutional layer. Also, the highest scoring networks used Convolution as the first layer which matches the usual approach to Machine Learning on image data.

These initial experiments verified that the evolution had been set up properly and gave me a better understanding of how it works.

No.	Type	Input size/channels	Output size/channels	Dependencies
0	Conv2D	270x50, 1 chn	134x2, 2 chn	-
1	Dense	268	4	0
2	Dense	272	2	0, 1

Table 4.3: *Architecture of a network evolved to recognize padded IMDb dataset examples with 59.852% accuracy using default parameters*

4.4 Sentiment analysis experiments

After verifying that evolution is configured properly I continued to perform some sentiment analysis experiments on IMDb dataset.

4.4.1 Sentiment analysis using default layers

At first I decided to try learning without adding any NLP-specific layer types. There were a few changes in regard to image processing. For instance, the Conv2D kernel's height was changed the size of an embedding vector. This allowed the layer to perceive groups of three words in one kernel which could help it take bigrams (like "not good") or trigrams into account instead treating tokens separately. After preprocessing the data I ran the experiment with the default configuration. At this point I fed the networks with all examples padded to a fixed size of 270 tokens. The best individual is shown in table 4.3.

The score of almost 60% over two classes indicates a little better accuracy than random guesses. The process again opted for processing the input with Convolutional layer capable of detecting local relationships followed by Dense layers collecting this information into the output.

I tried to increase the accuracy by testing various other values of parameters. Again I increased multiplier parameters and reduced the size of networks. Moreover, I followed the advice of my co-supervisor and increased the mutation probability to avoid stalls in evolution and enable more independence on the initial populations. I also increased the FP individual size (subset size) to give the networks more training examples. To compensate for the increase in the number of computations needed, I reduced the FP population size as big subsets have a higher probability of being representative anyway. I also reduced the number of training iterations.

Also, I noticed that no short networks (e.g. one or two layer networks) had been created. Normally, this should be possible even after choosing a high value for the parameter "Main individual size". The layers encoded in the genotype are connected in such a way, that some of them can be excluded from the resulting architecture (by not having a connection with the output layer). In other words, this parameter "Main individual size" determines an upper bound on the number of layers and affects the typical size of a network, but should not impede creation of small networks.

I noticed that this lack of small network was caused by the framework's code responsible for assuring that the network's input and output sizes match the dataset requirements. To ensure that, framework often added additional layers at the beginning and end of the network

No.	Type	Input size/channels	Output size/channels	Dependencies
0	Dense	13500	1142	-
1	Conv1D	1142, 1 chn	1140, 1 chn	0
2	Dense	1140	2	1

Table 4.4: *Architecture of a network evolved to recognize padded IMDb dataset examples with 65.28% accuracy using modified parameters*

that were not encoded in the genotype. Apart from that, there was no possibility of omitting all intermediate layers and connecting network's input to output with a single layer. In my experiments I wanted to include the possibility of evolving small networks, so I changed these two behaviours.

After all, I managed to achieve the best scores for the following parameters:

- Dense size multiplier: 1000
- Convolution size multiplier: 20
- Main mutation probability: 0.1
- Main individual size: 4
- FP individual size: 10000
- FP population size: 4
- Training iterations: 3

The resulting architecture is shown in table 4.4. The accuracy increased and the evolution decided to use the possibility of creating bigger Dense layers. Apart from this chosen network I checked that very short networks are created as well which confirmed that my code changes influenced the evolution possibilities.

4.4.2 Sentiment analysis using RNN

After these first experiments I decided to incorporate networks that are better suited for this task. At first, I tried using a regular unidirectional RNN "blindly", i.e. without any enhancements. It was supposed to receive a tensor of a fixed size just like any other layer type. In this case, I decided to interpret the "Extra" value of a gene as a factor influencing the hidden size of an RNN. Apart from that, I introduced a "RNN size multiplier" parameter in the configuration that worked in pair with the "Extra" gene value in a similar way as in Dense and Convolution layers.

The evolution process correctly created networks like the one presented in table 4.5. These networks were not able to learn from the dataset - the accuracy for all of them was more or less 50%, the score of random guesses. This might have been due to the vanishing gradient problem common to RNN, padding all examples to a fixed length or to the lack of enhancements that are often applied when tackling this sort of tasks.

No.	Type	Input size/channels	Output size/channels	Dependencies
0	RNN	270x50	256	-
1	Dense	256	2	0

Table 4.5: *Architecture of a network using unidirectional single-layer RNN created by evolution on padded IMDb dataset*

To address that I introduced a number of improvements:

- LSTM instead of a regular RNN,
- Two hidden layers instead of one (Multi-layer RNN),
- Bidirectional layer instead of unidirectional,
- Dropout regularization,
- Feeding unpadded sentences to the network.

The first four changes did not require a lot of work. Apart from passing the right values and flags to the constructor, I only had to adjust the output so that it took only the last hidden layer. The output shape was set to (hidden_dim, 2) because of the two-directional hidden state.

The last point, however, was quite difficult to achieve. Instead of padding all examples to a fixed length during preprocessing, I loaded the original dataset using some Pytorch tools on the run. I exchanged the default Sampler and Iterator with a BucketIterator which generates batches of sequences of similar length. This minimizes the amount of padding. I also extracted a vector of real lengths of sequences before padding. Then, I fed these batches into the LSTM, together with the vector storing their real lengths. This enabled the layer to process only the meaningful, non-PAD tokens of sequences.

This change meant that I had to introduce one more important condition: ensuring that only LSTM can appear in the architecture as the input layer. This is because the input of this layer has a very specific meaning and characteristics - it is supposed to be a sequence of varying length. Otherwise, the source sequences could not be of varying length, because for example a Convolutional layer expects a fixed size.

I decided to use the following parameters in order to give the networks an opportunity of learning on as much data as possible by training them on whole dataset. This meant that coevolution degenerated into a classic evolution - only the main population was evolved:

- FP population size: 1
- FP individual size: 24000
- Main individual size: 4
- RNN size multiplier: 250

No.	Type	Input size/channels	Output size/channels	Dependencies
0	LSTM	(Seq. length)x50	210x2	-
1	Dense	420	2	0

Table 4.6: *Architecture of a network using bidirectional multi-layer LSTM created by evolution on unpadded IMDb dataset*

These upgrades yielded far better results - the classification accuracy reached 72.54% for the individual described in table 4.6. It meant that networks that were capable of learning on IMDb dataset had been constructed.

Nevertheless, the evolution did not show almost any progress over time, even after trying out various configurations of parameters. Possibly it is difficult to improve an architecture starting with a LSTM layer by adding new layers afterwards. This would mean that the only way of evolving would be by changing the "Extra" parameter influencing the LSTM hidden layer size. In a way, this type of evolution has actually happened as the networks with extremely low hidden state size did not achieve good results. However, a bigger LSTM layer was often created already in the first evolution iteration, leaving little space for improvement in the following iterations.

4.5 Text summarization experiments

After learning how the evolution process works and investigating its capabilities in an easier NLP task I finally proceeded to design a solution for text summarization. In the first section I will analyse some examples of how the networks try to solve this problem and what challenges they did and did not manage to overcome. In the second section, I will describe my configuration for experiments, analyse constructed networks and their quantitive scores.

4.5.1 Qualitative analysis

The models that I tested during my experiments had shown various levels of capability of summarizing. Besides measuring these abilities with metrics like ROUGE or BLEU, it is worth analysing particular examples to see how these models approached the problem and evaluate their usability in real applications.

Out of the models that actually showed any progress during training, the worst ones created summaries like example 1 in table 4.7 for all the input texts. My interpretation is that such models were able to notice that the word "great" occurred frequently in reference summaries and tried to maximize their score by using a lot of it. This strategy allowed them to obtain scores higher than random word picking. The provided example is such a case - the model increased its score because the word "great" was present in the reference summary. The output, however, depends on the general characteristics of the dataset rather than on the particular input.

The models that were slightly better still repeated a word instead of constructing phrases, but were able to take input sequence into account, like in the example 2. The model must have noticed that the most frequent words from a given text have a high chance of appearing in

#		
1	Original text	*My two kitties differ on many of their food preferences, and we are also mindful of their health and so we try to limit their fish intake. So, we are grateful that we found the wellness turkey recipe. It's something that they both absolutely love and something that we feel good about feeding them.*
	Reference summary	*Great kitty food*
	Generated summary	*great great great great great great great great great great great*
2	Original text	*I had had this at Chicago airport at the cafe and this can of soup is close to the fresh soup. My kid also enjoys this soup too.... Recommend this product since its not the typical soup out of the can. Surely would like to sample the other soups from wolfgang puck.*
	Reference summary	*close to the fresh soup served wolfgang puck s cafe*
	Generated summary	*soup soup soup soup soup soup*
3	Original text	*If you are a Chai Latte fan, you must try this product! I 'spoil' myself with a cup whenever I'm off a day from work.*
	Reference summary	*this is a must have in my house*
	Generated summary	*chai latte chai latte latte latte latte*
4	Original text	*A friend sent me a package as a 'welcome home' gift for our new Burmese kitten...and these treats were a raging success. ALL of my Burmese love them, so do both Siamese! Since our new kitten seems to prefer only dry food, I use these treats (broken up) as an extra boost for his diet. As an added bonus, they already have the consistency of horked-up dried cat puke, yum!*
	Reference summary	*yum love them*
	Generated summary	*great product*
5	Original text	*These are by far the most addictive sunflower seeds in the history of mankind. I can't get enough of these things...seriously!*
	Reference summary	*the best snack addiction i have ever had*
	Generated summary	*the best seeds ever*

Table 4.7: *Chosen examples of summaries generated by various models constructed during experiments. (part 1)*

the summary as well, usually being the product's name, an important property or adjective. Here it used word "soup" which appeared four times in the text (and additionally one time as "soups") and it has been rewarded because of this word being present in the reference summary.

A more sophisticated version of this is the example 3, where two new abilities can be noticed. Firstly, the model managed to use a bigram "chai latte" instead of a single word. Moreover, it did not choose it based on a simple frequency in the input text. Instead, it managed to use its general knowledge to infer that this bigram is more important than others. It might have learned which particular words in vocabulary refer to products. Another possible explanation is that words from input text that are less frequent in the whole model's vocabulary tend to be more important.

While these achievements are promising in terms of understanding the input, these sequences can hardly be interpreted as summaries, because they do not form grammatical sentences. What was crucial to overcome this problem, was the ability to learn some phrases that often appear in the reference summaries. A basic example of this is the summary 4, where the model simply used a common phrase "great product".

Such summaries are often enough to be considered "correct", i.e. to be grammatical, concise, logically coherent and consistent with the meaning of the input. However, they are also quite generic - the phrase "great product" could be used to summarize any positive review (which form the strong majority of the dataset). To obtain summaries that are more specific and convey more information from the input, the models had to combine this ability of phrasing with the skill of extracting subjects and features from the text.

Summary 5 is an example of this. In order to construct it, the model had to understand that "seeds" is the reviewed product, that the review is very optimistic, that the phrase "the best X ever" can be used to express this optimism and that the product name should be put in the "X" position. Yet another impressive case is the summary 6 in table 4.8, where the model learned how to use a phrase "best X hands down".

The models did not just learn a couple of adjustable phrases that in general meant a positive opinion anyway. There were a number of summaries like example 7, where the model was able to inform who had an opinion on the product, and not only if they liked it or not. Moreover, it was able to extract it even though there was a lot of other information about dog's health and rationing. In example 8, the recommendation about getting the product as a gift was retrieved.

Unfortunately, even the best models capable of constructing these summaries also generated a lot of incorrect ones. They still struggle with some problems of the lesser-developed models described above, by producing looped sequences like example 3, or appending repetitions to otherwise correct summaries like in example 8.

Generic summaries like "good product" or "not what i expected" are frequent as well; they seem to be used as a default summary when the model has problems with understanding the text. This alone would not be an error - the opinions would just not very detailed for complicated texts. A bigger problem arises when the model does not recognize the sentiment correctly, applying one of the default summaries of the wrong sentiment, like in example 9. This could possibly be fixed by balancing the number of positive and negative training examples, using the "Score" column of this dataset.

6	Original text	*These bars are THE BEST! They are my go-to bar whether it be for a quick breakfast or a guilt free snack. Tasty and filling and the extra protein is a bonus. Try them but be forwarned.......they become a craving quickly.*
	Reference summary	*delicious and multi purpose*
	Generated summary	*best protein bar hands down*
7	Original text	*Our dog, a 10 year old bichon frise, goes wild when he sees these coming, so the flavor must be pretty great; he is not a huge fan of dry dog treats usually. After his recent difficulties with a teeth cleaning at the vet, I resolved not to have him go through that again, so it is a good thing to have the extra dental shield ingredient that protects teeth from tartar build up. There are also a lot of vitamins in the biscuits. The recommended feeding is 2 biscuits twice a day, which I think is way too much. This would be too filling and cause the dog to not eat enough of its dog food. If 4 biscuits per day are needed in order for the dental benefits to be realized, that is not good. Overall, I would use these biscuits several times a week to possibly give the dental benefits, but not as often as recommended. :)*
	Reference summary	*our dog is crazy about these*
	Generated summary	*my dog loves these*
8	Original text	*Perfect for a small gift topper or as a part of a gift bag or set! Just the perfect amount to make them enjoyable but not too much to feel like it's a waste if a flavor was... Surprising. Great fun for everyone.*
	Reference summary	*Cute!*
	Generated summary	*perfect gift gift*
9	Original text	*I usually use canned refried beans when I am in a hurry with Mexican dishes, or if I have the time..make my own from scratch. This product is AWWWFULL. They look great, but have a very funny, very strong, off-putting taste. Not sure what it is but will be returning the remainder of the order...ruined my entire dinner..YUK!!!!*
	Reference summary	*ugggh horrible*
	Generated summary	*great product*

Table 4.8: *Chosen examples of summaries generated by various models constructed during experiments. (part 2)*

10	Original text	*This drink is AWESOME! It tastes like cherry Amp but it has a bit more of a kick. I really enjoyed it. The only problem I had was the shipping cost was outrageous. Paid $50 for 2 of these drinks and they're only 8oz. Still good though =)*
	Reference summary	*BEST ENERGY DRINK I'VE EVER HAD!*
	Generated summary	*tastes like rubber*
11	Original text	*I got the same product at my local grocery store for $9.99. Much cheaper - what's going on Amazon? You aren't even offering a bulk price. Great product, my dogs love them too, but can be found much cheaper elsewhere.*
	Reference summary	*overpriced*
	Generated summary	*great price for price dogs love*

Table 4.9: *Chosen examples of summaries generated by various models constructed during experiments. (part 3)*

Sometimes the difficulty of the text confuses the models so that they produce serious errors and hallucinations like in example 10 in table 4.9. Here the error might have been caused by the fact that a big part of review was a negative opinion about the shipping cost, obscuring the general positive attitude. It hallucinated the word "rubber", information that was not present in the original text.

The models also have problems with understanding some concepts, e.g. prices. Attempting to summarize such information often ended up in generating text that was either inconsistent with the original meaning or even grammatically nonsensical, like in example 11.

4.5.2 Quantitive analysis

As mentioned earlier, I used a Seq2seq model with attention. It operates on an input of sequences of varying length, so similarly to LSTM solution in sentiment analysis, it had to be the first layer of the network. Additionally, the output of text summarization also has the same characteristic, which meant that Seq2seq model had to be the last layer as well. This led me to fixing the network architecture to be simply a single Seq2seq model and operate on its properties.

At first I assigned the "Extra" gene to influence the hidden size of the model together with "Seq2seq size multiplier". The rest of properties were fixed to the following configuration:

- Number of hidden layers: 1
- Bidirectional: Yes
- Dropout rate: 0.25

I ran an experiment with the following configuration (rest of parameters set to default):

Property	Value
Hidden size	456
Number of hidden layers	1
Bidirectional	Yes
Dropout rate	0.25

Table 4.10: *Architecture of a Seq2seq network created by evolution on Amazon Fine Foods Dataset with parametrized hidden size.*

Property	Value
NLL Loss	2.08
BLEU	0.1945
ROUGE-1	0.1321
ROUGE-2	0.0289
ROUGE-L	0.1321
F-Score	0.1396

Table 4.11: *Scores achieved by a Seq2seq network created by evolution on Amazon Fine Foods Dataset with parametrized hidden size.*

- Main individual size: 1
- Seq2seq size multiplier: 512

The generated models started to learn, the best reaching the Negative Log Likelihood loss of 3.74. However, this was not enough to create meaningful summaries. They were mostly similar to example 1 in table 4.7. The reason why the models did not learn anything better lies in the fact, that the loss did not seem to stop dropping before the end of training.

Therefore, I tried to repeat the same experiment, but changing the following parameters:

- FP individual size: 50000
- FP population size: 4

This yielded much better results. The loss decreased to 2.0-2.5 and the summaries started to be correct and meaningful. The architecture is shown in table 4.10 and the metrics scores in 4.11. The most important score that I will take into account in further comparisons is the F-Score as it combines both precision and recall.

In order to give more possibility of manipulating the networks to the evolution process, I introduced three more genes that influenced the following hyperparameters:

- Extra_layers - Number of LSTM hidden layers, range 1 - 2
- Extra_bidirectional - Unidirectional/bidirectional LSTM switch
- Extra_dropout rate - Dropout rate, range 0.0 - 1.0

Additionally, I further increased the size of the training subsets:

Property	Value
Hidden size	388
Number of hidden layers	2
Bidirectional	Yes
Dropout rate	0.08

Table 4.12: *Architecture of a Seq2seq network created by evolution on Amazon Fine Foods Dataset with parametrized hidden size, number of layers, bidirectionality and dropout rate.*

Property	Value
NLL Loss	1.92
BLEU	0.1704
ROUGE-1	0.1513
ROUGE-2	0.0083
ROUGE-L	0.1513
F-Score	0.1531

Table 4.13: *Scores achieved by a Seq2seq network created by evolution on Amazon Fine Foods Dataset with parametrized hidden size.*

- FP population size: 1
- FP individual size: 300000

This experiment gave the best results, the summaries were often accurate, like examples 6-8 in table 4.8. The best networks achieved losses of 1.9-2.0. The architecture that scored the highest is shown in table 4.12. Bidirectional networks with two layers have been chosen which confirms the usual approach to text summarization. Relatively low dropout rate was chosen which suggests that overfitting is not a big problem.

I was curious if increasing the possible values of hyperparameters beyond the usually used range could result in an increased score. The range of dropout rate was already the biggest possible so I changed the ranges of LSTM hidden layer size and the number of hidden layers, by setting the following configuration:

- Seq2seq size multiplier: 1024
- Extra_layers range: 1 - 4

This experiment, however, did not bring better results. Networks with big hidden layer size values like the one described in table 4.14 achieved very low scores, described in 4.15. Therefore, I concluded that bigger hidden layer sizes do not suit this dataset and task well.

Similarly, increased number of hidden layers did not bring improvement. The network's architecture is shown in table 4.16 and the results in table 4.17.

These experiments showed that increasing the size of a network does not always result in better scores. Moreover, some of these networks required much more computations to be trained, which is also often an important factor.

Property	Value
Hidden size	1010
Number of hidden layers	2
Bidirectional	Yes
Dropout rate	0.09

Table 4.14: *Architecture of a Seq2seq network created by evolution on Amazon Fine Foods Dataset with big hidden layer size.*

Property	Value
BLEU	0.0085
ROUGE-1	0.0083
ROUGE-2	0.0
ROUGE-L	0.0083
F-Score	0.0065

Table 4.15: *Scores achieved by a Seq2seq network created by evolution on Amazon Fine Foods Dataset with big hidden layer size.*

Property	Value
Hidden size	292
Number of hidden layers	4
Bidirectional	Yes
Dropout rate	0.06

Table 4.16: *Architecture of a Seq2seq network created by evolution on Amazon Fine Foods Dataset with increased number of hidden layers.*

Property	Value
BLEU	0.1142
ROUGE-1	0.0941
ROUGE-2	0.02
ROUGE-L	0.0941
F-Score	0.0922

Table 4.17: *Scores achieved by a Seq2seq network created by evolution on Amazon Fine Foods Dataset with increased number of hidden layers.*

Chapter 5

Conclusions

In this chapter I conclude the findings that arise from the results of experiments. I list the limitations that influenced my workflow and describe what possibilities should be explored in future research.

5.1 Research objectives

In the introduction I posed a couple of questions that guided my research.

Firstly, I found out that the evolution of neural networks can be used to develop neural networks for NLP. Given the layers suitable for these tasks it is capable of constructing the correct architecture. It also selects the hyperparameter values that give the best results.

When it comes to text summarization, the networks capable of generating good summaries were also created. The hyperparameters chosen by the process match the ones that are usually used when solving such tasks (2 hidden, bidirectional layers of size 128-512). This confirms that the process could be used to optimize hyperparameters for a given dataset without much manual work. The quality of these summaries met my expectations - they struggled with the same challenges as the state of the art solutions - redundancy, consistence with the source text and occasional hallucinations.

Nevertheless, the coevolution does not provide as much advantage as it could potentially bring in other fields. For instance, some image recognition solutions described in Szegedy et al. [2016] use architectures with a few hundred convolutional layers. In this case, each layer works in the same, uncomplicated manner. They can be easily stacked onto each other, allowing many configurations. It is the order and the sizes of layers that cause the network to provide valuable results - and these are the features that could be easily manipulated by automatic evolution, greatly reducing the amount of manual researchers' work.

Meanwhile, the solutions that provide the best results for NLP tasks are often composed of just a couple of components, like LSTM and attention. These are carefully crafted by researchers with a specific goal in mind and cannot be designed automatically. Evolution can only influence a couple of parameters, like layer's hidden size or the number of uniform hidden layers. This means that evolution helps with the task of hyperparameter optimization for a given network, but can give much less help in network architecture design.

The last question asked if engineers who have little expertise in the field of a given Machine

Learning task can use coevolution to design good networks. This could seem like a logical consequence of the fact, that this process can help finding the best order of layers or values of their hyperparameters. It can be true, but only for tasks which use common layer types. In other words, it could be true if the layers defined by default in the framework are useful in solving the given task. In my case, I needed to manually and intentionally introduce domain-specific solutions like Seq2seq as new layer, which required learning about them beforehand. This means that the tool can help an inexperienced user find the best network for a new dataset, but not for a new task.

Apart from the scientific objectives, I also adopted some personal ones. I can definitely assess that I learned a lot during this time. This concerns both the knowledge in the field of Machine Learning, NLP and evolution as well as skills in specific tools like distributed systems, queuing managers, computational environment, Python programming language, ML libraries, Linux operating system, databases and LaTeX. I will certainly use and deepen this knowledge because I plan to pursue a career in Machine Learning.

5.2 Limitations

Along the course of my study I encountered many issues that had an impact on my workflow.

My work relied on Pytorch-dnnevo framework. While it allowed me to perform the evolution experiments without having to write the whole mechanism myself, it also came with a couple of drawbacks. Most importantly, I had to design a way to set it up in a computational environment which took me a lot of time. I described this process in section 4.2. Also, even though my co-supervisor explained how coevolution works, I still had to understand the code without documentation in order to be able to modify it. For the same reason it was quite difficult to correct fix whenever they appeared. The evolution process needs to be able do handle many configurations of network architectures and experiments. This means that it is prone to errors, which can appear at any point of the experiment, causing it to be stopped.

Additionally, my level of experience in both the research of the field and in used tools further enlarged the consequences of these issues. I dedicated a lot of time trying to find the right computational environment, resolving errors and understanding the details of concepts of neural network architectures and text processing techniques. This left me with little time for performing the experiments.

5.3 Future research

In the following research it would be best to try to overcome the disadvantage of coevolution of neural networks that I pointed out earlier. While it is possible that this is a general property of coevolution that is impossible to eliminate, it must be noted that it could also be related to the specific implementation of Pytorch-dnnevo framework or to my decisions.

The implementation focuses on manipulating the general network architecture (number of layers and connections between them, etc.), which is well suited for deep neural networks. By default, it does not influence the internal properties of each layer. I only introduced the possibility of tweaking some of these properties, for instance decide if a given LSTM

should be bidirectional, how many hidden layers should it contain and what hidden layer size and dropout rate should it use. This could be driven further to tweak other properties like the usage of attention or other mechanisms introduced by researchers. Alternatively, Seq2seq model could be broken down into smaller layers, allowing the environment to insert additional layers between encoder and decoder. Yet another idea would be to allow the environment to stack multiple Seq2seq layers on top of each other which could possibly be used by the network to generate intermediate, medium-length summaries.

Besides, it would be valuable to investigate how the evolution would behave if other layer types and architectures used in NLP would be introduced, like the Transformer model. Testing using other accuracy metrics and datasets would be interesting as well.

In fact, I am going to continue this research at my home university, where I have three more months to develop it. As I have already overcome the difficult configuration issues, I hope to be able to quickly test some of the ideas described above. This will hopefully discover new ways of evolving neural networks for text summarization.

Bibliography

Bahdanau, Cho, and Bengio. Neural machine translation by jointly learning to align and translate. 2014. URL `https://arxiv.org/pdf/1409.0473.pdf`.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *ArXiv*, abs/1611.02167, 2017.

Bohm, Gao, Meyer, Shapira, Dagan, and Gurevych. Better rewards yield better summaries: Learning to summarise without references. 2019. URL `https://arxiv.org/pdf/1909.01214.pdf`.

Chen and Bansal. Fast abstractive summarization with reinforce-selected sentence rewriting. 2018. URL `https://www.aclweb.org/anthology/P18-1063.pdf`.

Feng, Guan, Li, Zhand, and Luo. Audio visual speech recognition with multimodal recurrent neural networks. 2017. URL `https://www.researchgate.net/publication/318332317`$_{A}udio_{v}isual_{s}peech_{r}ecognition_{w}ith_{m}ultimod$

Max Grusky, Mor Naaman, and Yoav Artzi. Newsroom: A dataset of 1.3 million summaries with diverse extractive strategies. 2018. URL `https://www.aclweb.org/anthology/N18-1065/`.

Gu, Lu, Li, and Li. Incorporating copying mechanism in sequence-to-sequence learning. 2016. URL `https://arxiv.org/pdf/1603.06393.pdf`.

Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. 2015. URL `https://proceedings.neurips.cc/paper/2015/file/afdec7005cc9f14302cd0474fd0f3c96-Paper.`

Baotian Hu, Qingcai Chen, and Fangze Zhu. Lcsts: A large scale chinese short text summarization dataset. 2015. URL `https://arxiv.org/abs/1506.05865`.

Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state of the art. 2017. URL `https://arxiv.org/abs/1704.05519`.

Khandelwal, Clark, Jurafsky, and Kaiser. Sample efficient text summarization using a single pre-trained transformer. 2019. URL `https://arxiv.org/pdf/1905.08836.pdf`.

Krizhevsky, Sutskever, and Hinton. Imagenet classification with deep convolutional neural networks. 2012. URL https://www.cs.toronto.edu/ kriz/imagenet$_{c}lassification_{w}ith_{d}eep_{c}onvolutional.pdf$.

Lixiang Li, Xiaohui Mu, Siying Li, and Haipeng Peng. A review of face recognition technology. 2020. URL https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=arnumber=9145558.

Chin-Yew Lin. Rouge: A package for automatic evaluation of summaries. 2004. URL https://www.aclweb.org/anthology/W04-1013.pdf.

Chin-Yew Lin and Eduard Hovy. Automatic evaluation of summaries using n-gram co-occurrence statistics. 2003. URL https://www.aclweb.org/anthology/N03-1020.pdf.

Liu, Lu, Yang, Qu, Zhu, and Li. Generative adversarial network for abstractive text summarization. 2017. URL https://arxiv.org/pdf/1711.09357.pdf.

Luhn. The automatic creation of literature abstracts. 1958. URL https://courses.ischool.berkeley.edu/i256/f06/papers/luhn58.pdf.

Zhihan Lv, Houbing Song, Pablo Basanta-Val, Anthony Steed, and Minho Jo. Next-generation big data analytics: State of the art, challenges, and future research topics. *IEEE Transactions on Industrial Informatics*, 2017. doi: 10.1109/TII.2017.2650204.

Maynez, Narayan, Bohnet, and McDonald. On faithfulness and factuality on abstractive summarization. 2020. URL https://arxiv.org/pdf/2005.00661.pdf.

Julian McAuley and Jure Leskovec. From amateurs to connoisseurs: Modeling the evolution of user expertise through online reviews. 2013. URL http://i.stanford.edu/ julian/pdfs/www13.pdf.

Minar and Naher. Recent advances in deep learning: An overview. 2018. URL https://www.researchgate.net/publication/323143191$_{R}ecent_{A}dvances_{i}n_{D}eep_{L}earning_{A}n_{O}vervie$

Kate Moran. How people read online: New and old findings. 2020. URL https://www.nngroup.com/articles/how-people-read-online/.

Kanchan M.Tarwani and Swathi Edem. Survey on recurrent neural network in natural language processing. *International Journal of Engineering Trends and Technology*, 48:301–304, 06 2017. doi: 10.14445/22315381/IJETT-V48P253.

Nallapati, Zhou, dos Santos, Gulcehre, and Zhiang. Abstractive text summarization using sequence-to-sequence rnns and beyond. 2016. URL https://arxiv.org/pdf/1602.06023.pdf.

title = Nancy Chinchor.

Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't give me the details, just the summary! topic-aware convolutional neural networks for extreme summarization. 2018. URL https://arxiv.org/abs/1808.08745.

Daniel W. Otter, Julian R. Medina, and Jugal K. Kalita. A survey of the usages of deep learning for natural language processing. 2019. URL https://arxiv.org/pdf/1807.10854.pdf.

Rush, Chopra, and Weston. A neural attention model for sentence summarization. 2015. URL https://www.aclweb.org/anthology/D15-1044.pdf.

Iqbal H. Sarker. Machine learning: Algorithms, real-world applications and research directions. 2021. URL https://link.springer.com/content/pdf/10.1007/s42979-021-00592-x.pdf.

Thomas Scialom, Paul-Alexis Dray, Sylvain Lamprier, Benjamin Piwowarski, and Jacopo Staiano. Mlsum: The multilingual summarization corpus. 2020. URL https://arxiv.org/pdf/2004.14900.pdf.

See, Liu, and Manning. Get to the point: Summarization with pointer-generator networks. 2017. URL https://arxiv.org/pdf/1704.04368.pdf.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. 2014. URL https://jmlr.org/papers/volume15/srivastava14a/srivastava14a.pdf.

Kenneth O. Stanley. Neuroevolution: A different kind of deep learning. 2017. URL https://www.oreilly.com/radar/neuroevolution-a-different-kind-of-deep-learning/.

Stiennon, Ouyang, Wu, Ziegler, Lowe, Voss, Radford, Amodei, and Christiano. Summarization with human feedback. 2020. URL https://arxiv.org/pdf/2009.01325.pdf.

Sutskever, Vinyals, and Le. Sequence to sequence learning with neural networks. 2014. URL https://arxiv.org/pdf/1409.3215.pdf.

Szegedy, Ioffe, and Vanhoucke. Inception-v4, inception-resnet and the impact of residual connections on learning. 2016. URL https://arxiv.org/pdf/1602.07261.pdf.

Tu, Lu, Liu, Liu, and Li. Modeling coverage for neural machine translation. 2016. URL https://arxiv.org/pdf/1601.04811.pdf.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. 2017. URL https://arxiv.org/pdf/1706.03762.pdf.

Pradeepika Verma and Anshul Verma. A review on text summarization techniques. 2020. URL https://www.bhu.ac.in/research$_{p}ub/jsr/Volumes/JSR_{6}4_{0}1_{2}020/48.pdf$.

Michael Volske, Martin Potthast, Shahbaz Syed, and Benno Stein. Tl;dr: Mining reddit to learn automatic summarization. 2017. URL https://www.aclweb.org/anthology/W17-4508.pdf.

Yakura, Shinozaki, Nishimura, Oyama, and Sakuma. Malware analysis of imaged binary samples by convolutional neural network with attention mechanism. 2017. URL https://www.researchgate.net/publication/318332317$_{A}udio_{v}isual_{s}peech_{r}ecognition_{w}ith_{m}ultimod$

Tom Young, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. Recent trends in deep learning based natural language processing [review article]. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018. doi: 10.1109/MCI.2018.2840738.