

ZHAW
ZURICH UNIVERSITY OF
APPLIED SCIENCES

BACHELOR-THESIS

SPRING 2017

**Building Conversational Dialog-Systems
using Sequence-To-Sequence Learning**

Authors:

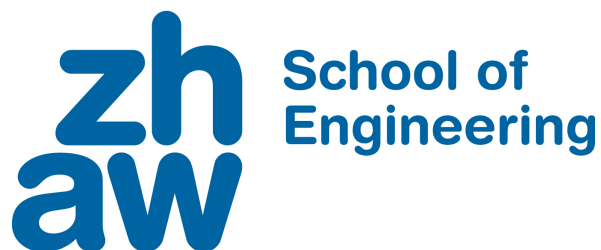
Dirk VON GRÜNIGEN
Martin WEILENMANN

Supervisors:

Dr. Mark CIELIEBAK
Dr. Stephan NEUHAUS

June 7, 2017

Zürcher Hochschule
für Angewandte Wissenschaften





DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:

Signature:

.....

.....

.....

.....

The original signed and dated document (no copies) must be included after the title sheet in the ZHAW version of all Bachelor thesis submitted.

Abstract

In this thesis, we investigate the construction of an end-to-end conversational dialog system based on recurrent neural networks and sequence-to-sequence learning. We use an already existing system as the reference architecture.

The main goal of the research is to investigate if a downsized model, trained on a single GPU, still exhibits a comparable performance compared to larger systems.

The analysis of the resulting models is done under several aspects. First, we look into the learning process. We show that the structure and linguistic nature of different datasets has a clear impact on the learning process of such models. Evaluations to assess the performance follow, including tests with a newly proposed metric based on the Sent2Vec library for measuring semantic similarity.

Our previous analysis revealed several problems our models exhibited. One of the main issues is that such models tend to generate generic responses which lead to deteriorating results. This does not match with our subjective impression that the models indeed become better over time. To find an explanation for this behavior we investigate how the language, used by the models, evolves over time. This analysis shows that language variety increases with training time and the occurrence of generic responses decreases.

A comparison with two other systems, namely CleverBot and the results from a paper by Vinyals and Le on “Neural Conversational Model” [1], follow subsequently. This comparison shows that our models have performance comparable to other systems, as long as the dialogs have limited complexity. We believe that the reasons for the comparatively worse performance of our models on complex dialogs include smaller model size, as well as shorter training time.

Although the results do not match our expectations, we are satisfied with the work. The models can, at least sometimes, generate meaningful dialogs, such as the response “i m a bot” to the question “what are you?”.

Zusammenfassung

In dieser Arbeit untersuchen wir den Aufbau eines End-To-End Dialogsystems mit Hilfe von Recurrent Neural Networks und Sequence-To-Sequence Learning.

Das Ziel dieser Arbeit ist es herauszufinden, ob ein kleines Modell, welches auf einer GPU trainiert wurde, vergleichbar gute Ergebnisse wie ein grösseres Modell hervorbringen kann.

Die Analyse der Modelle erfolgt unter mehreren Aspekten. Als Erstes analysieren wir den Lernprozess. Dabei zeigt sich, dass die Struktur und der sprachliche Ursprung der Daten einen spürbaren Einfluss auf den Lernprozess der Modelle haben. Anschliessend evaluieren wir die Leistung unserer Modelle, unter anderem mit einer neu vorgeschlagenen Metrik basierend auf der Sent2Vec Bibliothek, um die semantische Ähnlichkeit numerisch zu messen.

Die Analysen zeigen, dass unsere Modelle einige Schwierigkeiten haben. Ein grosses Problem ist, dass solche Modelle dazu tendieren, generische Antworten zu erzeugen, was zu einer Verschlechterung der Messwerte führt. Dies stimmt aber nicht mit unserem subjektiven Eindruck überein, dass die Modelle mit der Zeit besser werden. Um dieses Verhalten erklären zu können, untersuchen wir, wie sich die von den Modellen verwendete Sprache im Laufe der Zeit entwickelt. Diese Analyse zeigt auf, dass die Sprachvielfalt mit fortlaufendem Training grösser wird und der Anteil generischen Antworten sinkt.

Anschliessend folgt ein Vergleich mit zwei anderen Systemen, nämlich mit Cleverbot und den Ergebnissen aus dem Paper “Neural Conversational Model” von Vinyals and Le [1]. Dabei zeigt sich, dass unsere Modelle vergleichbar gute Ergebnisse wie die anderen Systeme erbringen, sofern die Dialoge nicht zu komplex werden. Die Gründe für die schlechteren Ergebnisse bei komplexen Dialogen vermuten wir in der Grösse der Modelle sowie der kurzen Trainingszeit.

Obwohl die Ergebnisse nicht ganz unseren Erwartungen entsprachen, sind wir insgesamt zufrieden. Unsere Modelle sind in der Lage, teilweise sinnvolle und interessante Antworten zu erzeugen, wie zum Beispiel “i m a bot” als Antwort auf die Frage “what are you?”.

Preface

First and foremost, we want to thank our supervisors Dr. Mark Cieliebak and Dr. Stephan Neuhaus for their ongoing support and guidance during the time we were writing this exploratory thesis. The weekly discussions were always a pleasure and helped us a lot to shape our ideas and goals. Jan Deriu also assisted us and we were always thankful for his advice. We also want to express gratitude to our families, colleagues and friends for their patience, support and encouragement to go on, as it was sometimes hard to keep up the motivation, especially in times where everything seemed to go wrong. Last but not least, we also want to thank the Institut für Angewandte Informationstechnologie from the ZHAW for providing us with the computational resources required to conduct the experiments.

Contents

1. Introduction	10
2. Related Work	12
3. Fundamentals	14
3.1. Definitions	14
3.2. Recurrent Neural Networks	15
3.3. Sequence-To-Sequence Learning	19
3.4. Performance Metrics	25
4. Software System	27
4.1. Requirements	27
4.2. Development of the System	28
4.3. Model Validation Checks	31
4.4. Web-UI	32
4.5. Scripts	32
4.6. Hardware	33
4.7. Operating System & Software Packages	33
5. Data	35
5.1. Datasets	35
5.2. Structure of the Raw Corpora	36
5.3. Preprocessing	38
5.4. Vocabulary and Coverage	40
5.5. Splitting the Datasets	42
5.6. Time-Lag Analysis OpenSubtitles	42
5.7. N-Gram Analysis	43
6. Methods	45
6.1. Architecture of the Sequence-To-Sequence Model	45
6.2. OpenSubtitles and Reddit Models	47
6.3. Hyperparameters	47
6.4. Training	48
6.5. Evaluation	48
7. Results and Discussion	50
7.1. Was the Training Successful?	50
7.2. Performance on Test Datasets	52

7.3. Development of Language Model	60
7.3.1. Uni- and Bi-Gram Distributions over Time	60
7.3.2. Language Diversity	64
7.3.3. Results	67
7.4. Comparison with Other Models	68
7.4.1. Cleverbot	68
7.4.2. Neural Conversational Model	75
7.4.3. Results	79
7.5. Beam-Search	80
7.6. Thought Vectors for Input Sequences	81
7.7. Soft-Attention	83
8. Conclusion	86
9. Future Work	89
Appendix	91
A. Neural Networks	92
B. Using the Software System	96
B.1. Download	96
B.2. Requirements	96
B.3. Structure of the Repository	98
B.4. Using the Scripts	99
B.5. Running Experiments	101
B.6. Web Frontend	105
C. Uni-Gram Distributions over Time	107
C.1. OpenSubtitles	108
C.2. Reddit	109
D. Results Compared with Neural Conversational Model	110
E. Beam-Search Results with OpenSubtitle	114
E.1. Results for "what year were you born?"	114
E.2. Results for "i wear a blue jeans. what color do my jeans have?"	115
E.3. Results for "what is the purpose of being intelligent?"	117
F. Soft-Attention Visualizations	119
Glossary	125
Acronyms	126
List of Tables	127
List of Figures	130

1. Introduction

The development of dialog systems or chatbots which are capable of passing the Turing test [2] is a long lived dream among the research community. Since the publication of the Turing test, a lot of efforts have been taken to achieve this goal, but none was so far able to¹, even though there were some serious contenders. The research area has gained a lot of attention in the past few years due to the technical advancements occurred over the last two decades. One of the big leaps in technology was the inception of machine learning, especially deep learning. The theoretical fundamentals have been available for a long time, but were not applicable due to the lack of computational power and amount of processed data required by such systems. However, this has changed in the past few years where deep learning had a huge influence on various different fields, for tasks such as image recognition, natural language processing or anomaly detection. They have proven that it is indeed viable to use this technology nowadays because of the exponential growth in available data and computing power.

In this thesis, we will try to develop a conversational dialog system comparable to the one published in the paper *Neural Conversational Model* by Oriol Vinyals and Quoc V. Le [1]. Our main motivation for building such a system is twofold: First, their system is an end-to-end system, meaning that the system as a whole can be designed and trained without the obstacle of managing several different subcomponents necessary for performing such a task (i.e. language understanding, interpretation and generation). Our second motivation lies in the sheer enthusiasm about the fact that the system presented in the mentioned paper, despite its simplicity, was able to learn to respond meaningful to important questions, such as:

Human: what is the purpose of life ?

Machine: to serve the greater good .

We were blown away when we first saw the machine answering in such a decisive and comprehensible way we would never have imagined possible.

Our goal is to build a system comparable to the system in the paper, but on a much smaller scale. Our aim is to implement a system which is capable of being trained on a single GPU, instead of requiring an entire server with all its RAM and CPUs. For this reason, we investigate if such a model can also be used if scaled-down significantly, namely half of the original size. We then use this scaled down version of this model to

¹<http://isturingtestpassed.github.io/>

train it on two separate datasets: One on the OpenSubtitles [3] and the other on the Reddit Comment Dataset², specifically using the comments from the subreddits *movies*, *television* and *films*. We have chosen the first dataset in order to evaluate if a smaller model can compete with the results from the mentioned paper. The main idea behind choosing the second dataset is that we want the resulting dialog system to be able to talk about movie-related topics.

The trained models are then analyzed in different dimensions. First, we are exploring the issue of performance metrics and try if a newly proposed metric based on the Sent2Vec [4] library is sufficient for evaluating semantic similarity in such a setup. We then analyze, how the models learn to talk and try to evaluate which aspects are important for doing so by analyzing the produced language models and compare these models to the language models found in the mentioned datasets. A comparison between our models and the model found in the paper and the CleverBot³ then follows. Finally, our goal is to analyze if beam-search helps to diversify the answers and if and how the soft-attention mechanism [5] can help the models to increase comprehension.

²<https://archive.org/details/2015-reddit-comments-corpus>

³<http://www.cleverbot.com/>

2. Related Work

The history of dialog systems and chatbots dates further back than most people would probably expect, as the first known dialog system, called ELIZA, was already published in the year 1966 by Joseph Weizenbaum [6]. Even though this system was pretty primitive from a today's standpoint - it used scripted responses instead of learning to talk - it was widely recognized and was considered one of the first contenders to pass the Turing test. One of its successors was then PARRY, which was a chatbot simulating a patient with paranoid schizophrenia [7]. An interesting note here is that it was the first actual computer program to almost pass a special variant of the Turing test as only 48% of the participants involved in the test classified it correctly to be a machine. It took many years and several other implementations of chatbots, such as Jabberwacky, Dr. Sbaitso, A.L.I.C.E and SmarterChild until today's highly sophisticated systems became available. The biggest problem with most of these earlier systems was that they did not actually learn anything, but were driven by rule-based frameworks instead. Maybe the best known, intelligent dialog system is the IBM Watson system [8], which has shown its performance several times by beating the world champions in the question-answering game *Jeopardy!*. It is built by combining several, complicated subsystems, each responsible for a certain task (e.g. language understanding, information retrieval, answer ranking). Since the rise of deep learning within the last century, ample of different ways to build dialog systems have been proposed.

During the research for our thesis, we found the paper “Neural Conversational Model” by Oriol Vinyals and Quoc V. Le [1], which immediately got our attention for several reasons: First, it was one of the few systems using an end-to-end approach, which means that a single closed system is responsible for the understanding and processing the input as well as for the generation of the responses. We found that an appealing property because it helps to lower the costs of management, design and integration. On the other hand, we were also interested because it not only uses a single neural network, but also a fairly new architecture, called *Sequence-To-Sequence* learning, or seq2seq in short. This architecture was proposed by several different researchers around the same time [9][10][11], mainly as an alternative for statistical machine translation. It was quickly discovered afterwards, that the proposed architecture could not only be used for machine translations, but also for various other tasks requiring to map input to output sequences, such as image captioning [12], syntactic constituency parsing [13] and dialog systems or chatbots [14][3]. There are also some unsupervised use-cases for seq2seq models, such as generating embedded representations of videos [15].

In this thesis, we want to evaluate if the results found in the *Neural Conversational Model* paper can be replicated by a much smaller system, or to be precise, a model which is half

the size of the original one. Our objective is to fit the entire model onto a single GPU for training and inference instead of an entire server with multiple CPUs as they did in the paper. We then try to analyze the development of the models over a prolonged time of training, as this model need a lot of time to train. As the last point, we are going to investigate how beneficial other extensions for the resulting models are, such as attention and beam-search.

3. Fundamentals

In this first Chapter, we lay out the fundamentals used in the rest of this thesis. At the beginning, we will introduce basic definitions such as *utterance*, *sample* and *dialog*. We will then follow up with an introduction into machine learning using a special variant of *Neural Networks* (NN), called *Recurrent Neural Network* (RNN). We show the basic principles behind it and explain problems this RNNs have in practice, and how they can be solved by utilizing other forms of RNNs. We then show how RNNs can be used to build models, called *Sequence-To-Sequence* (seq2seq) models, which are capable of learning and using a language in a conversational context.

The following introduction only covers a small part of the spectrum of possibilities with regard to the tasks these models can perform. We are restricting our explanations to the supervised learning use-case and ignore the unsupervised ones, even though they have a wide area of applications (e.g. dimensionality reduction, regression). A basic introduction into the principles of neural networks can be found in Appendix A.

3.1. Definitions

Utterance An *utterance* is a single statement from one of the participants in a dialog. This means that an utterance can be either the initial statement or the response to this statement. An utterance may consist of several sentences, but they are always handled as if these sentences were uttered in one unbroken sequence by one participant without the second one interjecting.

Sample A *sample* is a combination of two utterances. The first is usually the utterance by the first participant, and the second one is the response to this first utterance by the second participant of the dialog.

Dialog A *dialog* is a conversation between two parties consisting of possibly multiple samples with two utterances each. An important distinction from how the term dialog is usually used in everyday life is, that we do not take the dialog context into account. This comes from the fact, that the model used in this thesis is not capable of handling context between multiple samples. We nevertheless use the term dialog to describe a string of multiple samples following each other.

The details on how dialogs are handled when training or doing inference with the model are described in detail in Chapter 6.

3.2. Recurrent Neural Networks

Recurrent neural networks (RNN) are a special variation of the NNs described in Appendix A. The main difference between them is, that RNNs have a recurrence built-in that allows them to adapt to problems which also have a temporal dimension and are dependent on data from different time steps to solve. We are also going into the problems such RNNs have due to their recurrence and how this problem can be solved by exploiting another form of RNN, called *Long Short-Term Memory Networks* (LSTM).

Operating Principles of RNNs One of the main restrictions of vanilla NNs is the following: assume a task in which the NN is conditioned to output a prediction on tomorrow’s weather given yesterday’s. Now, if one would use a vanilla NN as described in Appendix A, the main restriction would be that the NN has to make its prediction solely based on the weather information of the day before. Such a model does not take into account, that weather is not only dependent on the weather of the previous day, but also on the days before. This could be solved by feeding, say, the weather of the last week to the network instead of just the weather of the previous day. But if new scientific evidence now shows, that weather is not only dependent on last week, but also on the last month, probably the last year, we quickly get into problems due to the sheer size of a NN performing such tasks, because the input size grows rapidly. Also, such a NN would still be static in the sense that one cannot simply change the time-window used to feed to the network. If one settles for one month, it will always be able predict the weather based on the last month, but not any different time-window, otherwise the NN has to be retrained using another time-window in order to work again as before. RNNs (see Figure 3.1) solve this problem by introducing a recurrence into the network, which allows it to exploit information not only from the current input, but also from inputs of the past. It does this by transferring a state, usually called the *hidden state*, through the recurrence between different time steps. In a more compact way, one could say that this recurrence allows RNNs to “exhibit dynamic behavior across the temporal dimension of the input data”.

Before explaining how this recurrence can be used to solve the weather prediction problem, let us first show the equations used for the forward propagation in a RNN with a single cell. A single layer in an RNN is called a *cell* and is basically a function which maps tuples of state and input to new tuples of a new state and an output $f: \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n \times \mathbb{R}^m$, where n signifies the size of the hidden state and m the size of the output of the cell.

The internal structure is similar to the one of NNs, except for the recurrence which we

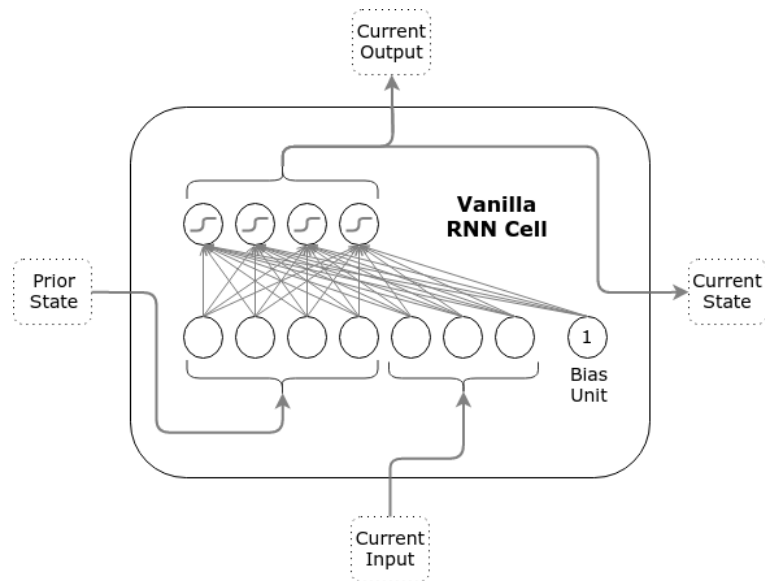


Figure 3.1.: Internal structure of a vanilla RNN cell.¹

are going to elaborate on below.

$$\begin{aligned} h_t &= \varphi_h(\mathbf{w}_h \cdot \mathbf{x}_t + \mathbf{u}_h \cdot \mathbf{h}_{t-1} + \mathbf{b}_h) \\ y_t &= \varphi_y(\mathbf{w}_y \cdot \mathbf{h}_t + \mathbf{b}_y) \end{aligned} \tag{3.1}$$

In the equations above, one can see different variables with different indices which we will explain briefly:

- x_t stands for the input at time step t .
- y_t stands for the output of the network at time step t .
- \mathbf{h}_t stands for the hidden state at time step t .
- \mathbf{w}_h , \mathbf{w}_y and \mathbf{u}_h are the weight matrices learned while training the model.
- \mathbf{b}_h and \mathbf{b}_y stand for the bias vectors used when computing the new hidden state or output respectively.
- φ_y and φ_h are the activation functions used to compute y_t and h_t respectively.

As seen above, at every time step t , the new hidden state h_t and the output y_t are computed. The value of y_t can be seen as the prediction of the RNN at time step t and the value of h_t is the value that is passed to the same cell in the next time step $t + 1$ for computing the next output y_{t+1} (and hidden state h_{t+1}). As depicted in Figure 3.1, the hidden state h_t is returned to the cell for computing the output y_{t+1} and h_{t+1} . This is what an RNN allows to exhibit behavior depending on data seen in past time steps: It can “remember” what it has already seen and store the information necessary for the prediction in the hidden state, which is passed along the temporal dimension when a RNN is run through multiple steps in time. To come back to our weather prediction

¹<http://r2rt.com/static/images/NH-VanillaRNNcell.png>

problem, the recurrence and the hidden state passed along it allows the cell to remember what weather it has seen in the past and adjust its future predictions accordingly. The recurrence also solves the problem of time windows of different sizes: Since the recurrence allows the model to store the required information in the hidden state at each time step and combine it with what it has already seen to make future predictions, it is not constrained to a fixed size of inputs required to make a prediction but instead uses all information it has already seen, no matter if the seen information is from the past five days or past five years, it can compress all this into the hidden state h_t .

While this is the solution for our problem of different window sizes, it is also a main bottleneck of the model: If one imagines, that the time window is past information is five years and we process these information on a daily basis, we would need to remember the information from around 1800 days. If we would now have 10 distinct features per day, this leads to 18,000 features to remember in total. Remembering means to compress the information into the hidden state, which is usually much smaller than 18,000 entries, reasonable values vary from 128 up to 4096 entries, depending on the structure and complexity of the problem. Another obstacle is, that the RNN cell has no way of forgetting information stored in the hidden state easily, as Equation 3.1 only adds an addition to the hidden state h_{t-1} . Of course, in theory it is still possible that the RNN learns to forget insignificant information, but this is a really hard problem in practice, which is why we will introduce yet another form of RNN cell, called *Long Short-Term Memory* cell that not only solves this problem, but also partially the problem of unstable gradients explained in the next paragraph.

Vanishing / Exploding Gradients The recurrent nature of RNNs can cause problems in practice, one of the most problematic is the vanishing gradients problem: While training such models, we condition the weight matrices \mathbf{w}_h , \mathbf{w}_y and \mathbf{u} via backpropagation by using gradient-descent in such a way, that the loss function is minimized with regard to these parameters for the given training samples. This means, that we have to compute the derivatives of the loss function by using the chain-rule $dz/dx = (dz/dy)(dy/dx)$. The entirety of all these derivatives w.r.t. to the parameters is called the gradient. Since the gradient also flows through the activation functions φ_h of the recurrence, these computations also include the derivatives of this function. Most of the commonly used activation functions today have codomains in \mathbb{R} and take on values in $[-1, +1]$ (e.g. \tanh). This means that multiplying many of this derivatives possibly leads to vanishing values. This has a severe effect on training the RNN: If the values are vanishing, there is a point after which the network stops learning. This is because the change in weights becomes so small that the predictions will not change. This problem becomes especially acute when we use RNNs (or very deep NNs) with large window sizes as they act the same way as deep neural networks because they are unrolled when backpropagating.

Another problem which can occur is the opposite of the explained vanishing gradients: Exploding gradients. We talk about exploding gradients if the norm of the gradient becomes unreasonably large, to the point where it hurts the learning process severely. The gradient can explode due to the fact that the errors in the network propagate through an extended time window. This propagation takes the form of a multiplication of multiple

Jacobian matrices. If the norm of this matrices is larger than one, then the gradient might explode, especially if the used time window is large. A simple solution to this problem is to use gradient clipping [16]. It solves the problem by clipping the norm value of the gradient if it exceeds a certain threshold.

The problem of vanishing gradients was first analyzed by Hochreiter in his diploma thesis [17]. An analysis and in-depth explanation of the exploding gradient problem can be found in [18]. A more formal explanation of the problem, with links to dynamic systems and formal proofs, under which conditions these problems occur, can be found in [16]. The details on how backpropagation with gradient-descent is adapted to RNNs and time-dependent problems, called *Truncated Backpropagation Through Time* or *TBTT* in short, can be found in [19].

Long Short-Term Memory Networks To solve the problem with vanishing gradients mentioned before, Schmidhuber and Hochreiter introduced an advanced version of a RNN cell, called *Long Short-Term Memory*, or LSTM in short [20]. The structure of this kind of cells is much more sophisticated than vanilla RNN cells, but it also solves the two most pressing problems: Vanishing gradients and the difficulty of forgetting stored information.

It does this by introducing the so-called *gates* into the cell, which are nothing else than small NNs which are responsible for deciding which information from the previous time step we are going to use from the old hidden state h_{t-1} , add to the new hidden state h_t and use for the computation of the output y_t . There are three different gates in an LSTM cell (as depicted in Figure 3.2):

- **Input gate:** Is responsible to decide which part of the input is interesting and should be used to update the hidden state h_{t-1} to become the new one h_t .
- **Forget gate:** Is responsible to decide which part of the hidden state h_{t-1} the cell is going to forget.
- **Output gate:** Decides which part of the hidden state h_t is used to compute the output y_t .

In theory, these gates do nothing more good than a plain RNN cell, as plain RNN cells are already Turing complete and hence can compute any desired function which is computable by a Turing machine [22]. However, in practice this structure solves all of the aforementioned problems to at least some degree.

First, let us look into the vanishing gradient problem: The problem occurred due to the fact, that the derivation of the activation function for the recurrence is used when calculating the updates of the weight parameters by performing backpropagation with gradient-descent. This can lead to vanishing gradients. The LSTM cell solves this problem by not applying an activation function on the recurrence, but rather using the identity

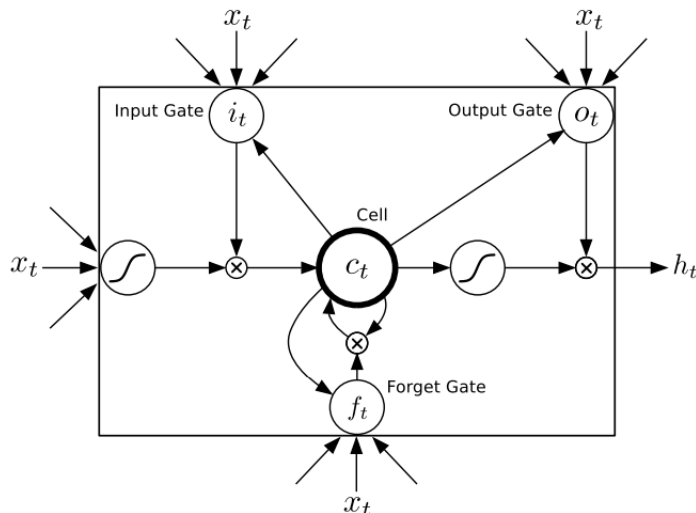


Figure 3.2.: Internal structure of a LSTM cell [21].

function $f(x) = x$ whose derivation is always 1. This practically solves this problem because the gradients do not vanish anymore.

The second problem is that an RNN cell has no explicit way of forgetting information stored in the hidden state. This is solved by the introduction of gates which allow the cell to decide at each time step t , which information to keep, forget and store based on the last hidden state h_{t-1} and the current input x_t . With this structures in place, the LSTM cell is able to track long-term dependencies much better than a vanilla RNN cell.

There are more details regarding the LSTM cell, like the forget bias and peephole connections, which can be looked up in an empirical study by Greff et al. [23]. There are even more architectures for RNN cells like Gated Recurrent Unit [24] and Convolutional LSTM [25], which we will not describe in detail in this thesis.

3.3. Sequence-To-Sequence Learning

The RNN and LSTM cells described in the preceding Chapter can now be used to build so-called *Sequence-To-Sequence* (seq2seq) models. They were first introduced by several people around the same time [9][10][11], mainly for usage in the context of machine translation tasks, but they can also serve as a generic model for learning to map arbitrary input to output sequences. They have shown, that such models can be successfully adapted to machine translation tasks and exhibit almost state-of-the-art performance. For more general information on where and how seq2seq models used we refer to Chapter 2. In the following paragraphs, we are going to explain the basic idea behind the model and how each of its parts, called encoder and decoder, work when applied to conversational modeling.

Model The basic idea behind the model is to separate the parts necessary to understand and parse the input sequence, called the *encoder*, from the part that is responsible for generating the output sequence, called the *decoder* (see Figure 3.3). Both the encoder and decoder can be any kind of RNN cell, be it GRU, LSTM or any other. The encoder is fed with the input sequence through multiple time steps, as an example one could say that for example each word of an input sentence is fed to the encoder, one per time step. Through this process, the encoder starts to build its internal hidden state, updates it every time a new word is fed and passes it along the recurrence to serve as the initial state of the cell at the next time step. After the input sequence is fully processed by the encoder, it then passes the information collected in the hidden state, in the context of seq2seq models also called *thought vector*, to the decoder cell. The decoder cell then uses this hidden state as its own, internal, initial state and starts to produce the output sequence, one token per time step, after its fed with a dedicated GO token (also depicted in Figure 3.3). The tokens are produced by feeding the final state of the decoder cell to a softmax classification layer which outputs a probability distribution over the words in the vocabulary. It then usually chooses the “best” token to output in a greedy fashion by simply selecting the token with the highest probability after the softmax layer has been applied. The details on how the decoding exactly works are described in the paragraph *Decoding Approaches* below. In general, the decoder tries to construct a sensible output at every time step, in our case this corresponds to the current input word, t given the hidden state from the decoder at time step $t - 1$ and the last output y_{t-1} .

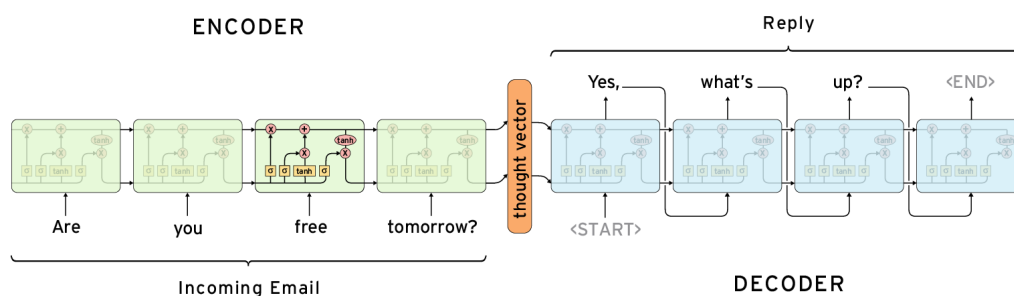


Figure 3.3.: Internal structure of a Sequence-To-Sequence Model.²

Such a model is also called an *end-to-end* model because it, being fully differentiable, can be trained by using backpropagation with gradient-descent with ease. The training works by having pairs of input-output samples which are then used to train the network and condition it to generate the expected output given the input sequence. This is done by feeding the network the expected words as inputs at each time step which allows for gradient-descent to be applied. Formally, if we are talking about language modeling, the model is conditioned to optimize the conditional probability $p(y_1, \dots, y_T | x_1, \dots, x_{T'})$, where T is the length of the input sequence and T' the length of the corresponding output sequence. As said before, the encoder cell produces a thought vector v , which is nothing more than its hidden state after it has processed the input sequence x_1, \dots, x_2 , and passes this to the decoder for generating the output sequence. The decoder is then trained to

²<http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/>

compute the probability of $y_1, \dots, y_{T'}$ given the thought vector v :

$$p(y_1, \dots, y_{T'} | x_1, \dots, x_T) = \prod_{t=1}^{T'} p(y_t | v, y_1, \dots, y_{t-1}) \quad (3.2)$$

Each of the terms $p(y_t | v, y_1, \dots, y_{t-1})$ in the product represents a distribution over all words of the vocabulary that is computed when applying the softmax layer at the end. What might be perplexing to the reader is that on the right hand side of the equation, the inputs x_1, x_2, \dots, x_T are left out. This has to do with the fact that the decoder produces its prediction solely based on the thought vector v produced by the encoder and the decoders previous predictions y_1, \dots, y_{t-1} (in a formulation without soft-attention, see below).

If we look at the example depicted in Figure 3.3, the generation of the output sequence is done as follows:

1. Feed the encoder the current input word x_t and update its hidden state accordingly. The output of the cell can be ignored.
2. Repeat step 1 until the inputs x_1, \dots, x_T are exhausted.
3. Pass the hidden state of the encoder cell to the decoder cell after the first has processed the last word x_T of the input sequence.
4. Initialize the decoder with the given hidden state and feed it the dedicated GO token.
5. Store the output token y_t of the decoder cell. After this, it depends on which decoding approach is used to generate the output sequence:
 - a) In case of a greedy decoder, simply feed the output y_t and hidden state h_t back in to the decoder cell to produce the next output token y_{t+1} and hidden state h_{t+1} .
 - b) In case of a beam-search decoder, select the N outputs y_1, \dots, y_N with the highest probability and store them together with the respective hidden states. Feed each of this outputs with the respective hidden state back into the decoder cell.
6. Repeat step 5 until the decoder either outputs an EOS token or the maximum length T' for the output is reached.

The concatenated outputs $y_1, \dots, y_{T'}$ are the output of the model. The exact decoding approaches are described in more detail in the *Decoding Approaches* paragraph below. Generally speaking, such models can be used for *any* sequence-to-sequence problem, not just language modeling, as long as it is possible to define a probability distribution over the output tokens y_t . Another fact worth mentioning is, that we described this model when using a single RNN cell for encoding and decoding, but it is indeed possible to use multiple cells for each of these parts.

Soft-Attention Mechanism Imagine you have been asked to solve the following simple task: Read a longer text about a certain topic. At the end of the text, there are several questions regarding this text and you have to answer them. How does a human approach such a task? At least a part of the people start by reading the text through one time and then start to answer the questions. A normal human cannot remember the whole content of the text (since it is a long one), so what it does is to focus on a single question and try to find the answer by revisiting the text. This essentially means that the participant in this task *focuses* their *attention* on single parts of the text instead of the text as a whole. This is basically what the *soft-attention* [5] mechanism is all about. Let us adapt the introductory example on how this behavior could be adapted to seq2seq models and how it helps to solve tasks.

As mentioned before, the main bottleneck of seq2seq models is that the encoder has to compress all the information it has processed into its thought vector. The thought vector is then passed to the decoder which tries to come up with a meaningful answer to the information stored in the thought vector. If a long text is compressed into the thought vector, this might not be an easy task to solve as the information has to be compressed too much. The attention mechanism helps by solving this issue by allowing the decoder to look at all thought vectors of the encoder at all time steps via a weighted sum.

We are going to describe the attention mechanism in more detail now and follow the explanations in [13] to do so. Formally, the attention mechanism works as follows: Consider that we already have all hidden states of the encoder $\mathbf{H} = \{h_1, \dots, h_T\}$ at the time we start decoding, where T stands for the last time step the encoder had to process an input word. We also have the hidden states of the decoder $\mathbf{D} = \{d_1, \dots, d_t\}$ (since attention is applied after the cell has predicted the current token at time step t). We use the same number of hidden states in \mathbf{H} and \mathbf{D} for the sake of simplicity, but it could be possible that there are a different number of time steps for the encoder and decoder. The attention vector d'_t is computed with the following equations:

$$\begin{aligned} u_i^t &= v^T \tanh(W_1 h_i + W_2 d_t) \\ a_i^t &= \text{softmax}(u_i^t) \\ d'_t &= \sum_{i=1}^t a_i^t h_i \end{aligned} \tag{3.3}$$

The index t stands for the current time step. The index i ranges from 1 to the number of steps we have computed in the encoder, which is equal to T in our case. The vector v and the matrices W_1 and W_2 contain the learnable parameters for the attention mechanism. The vector u^t is of length t and its entries signify, how much attention should be put on the hidden state h_i of the encoder. By applying a softmax layer on the vector u^t , we turn it into the probability distribution a^t over all hidden states \mathbf{H} from the encoder. The final computation of the d'_t is then done by computing the weighted sum between the weight vector a^t and the respective hidden state h_i from the encoder. This vector d'_t , also called

context vector, is then concatenated together with the current hidden state of the decoder d_t to become the new hidden state with which we go on predicting the next output token at time step $t + 1$.

The weights W_1 and W_2 , as well as the values in the vector v , are learned while training the model. This means, that ability to be “attentive” is something the model has to learn and hence cannot be added after the training. This also means that choice on what the model attends is not only dependent on the current sample, but also on the task in general. You find an example of the attention weights, visualized with a heat map in the Figure 3.4.

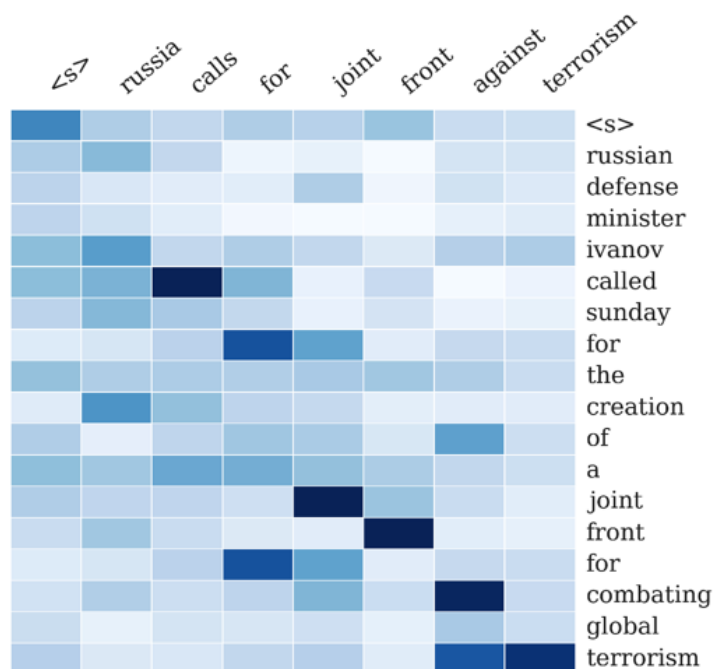


Figure 3.4.: Example on how attention can be visualized using a heatmap [26]. The input sequence is on the y-axis and the respective response on the x-axis. Each column in the heat-map can be interpreted as the probability distribution over the thought vectors of the encoder which the decoder accesses at each decoding step.

The name *soft*-attention comes from the fact, that this kind of attention mechanism is fully differentiable and hence can be simply plugged into any existing NN architecture. The other kind of attention is the so-called *hard*-attention mechanism which in contrast samples a random thought vector of the encoder and because of that, it is not fully differentiable and hence cannot simply be added to an existing system that uses back-propagation and gradient-descent without modifications.

We have only explained on how soft-attention can be used in a context where we process language. However, the attention mechanism itself originated from the computer vision area [27][28][29] and has a wide range of applications there [30][12][31]. The

mechanism can be adapted to any kind of problem and model, not just computer vision or NLP, as long as the model has any way of accessing states related to the input data.

Decoding Approaches In the following paragraph, we are going to elaborate on two decoding approaches that we are going to use in our experiments: *greedy* decoding and *beam-search* decoding.

Greedy decoding is the simpler of the two variants. When using it, the output y_t of the model using it is simply the token with the highest probability after the softmax layer has been applied on the last hidden state h_t of the decoder at time t . It then goes on by feeding in the hidden state h_t and the last output y_t to produce the output y_{t+1} for the next time step $t + 1$. This variant is really simple to implement, but also has its flaws: Since the decoder is working in a local and greedy fashion, it does not know beforehand what kind of sentence it would predict best, and hence has to stick with local optima on each time step t by returning the word with the highest probability after applying the softmax.

Beam-search tries to fix or at least minimize this problem by not only considering a single candidate sequence but instead focuses on the m different sequences that are most promising. The variable m is called the *beam width* and signifies how much possible candidate sequences are under consideration. The approach itself works by doing the usual computation and process the input sequence with the encoder cell at the beginning. The last hidden state of this encoder is then passed to the decoder which starts to do the beam search. It begins by using the thought vector (and the GO token) to predict the first word of the output in the same way as the greedy decoder does. But now, instead of simply choosing the next token to be the one with the highest probability, the decoder stores the m tokens with the highest probabilities. In the next step, the decoder is now fed with all of the m tokens produced in the step before and their respective hidden states. After this second step, the decoder has to *prune* the beams by deciding which of the m sequences it wants to keep and which it wants to discard. This is done because after the second step, there are actually m^2 candidate sequences combined in all beams, which violates the rule that there may be at most m beams. The m^2 candidate sequences are ranked by the sum of the log-probabilities of all the predicted words at each time step in every beam. Instead of using the plain probabilities from the softmax layer, the log-probabilities are used to ensure that the computation is numerically stable, as the multiplication of a lot of small values can lead to instabilities. After the ranking, only the m most promising beams are kept and all others are rejected. This process is then carried out until we reach an EOS token or the maximum number of allowed time steps in the decoder is reached for all beams. To decide, which is the best predicted sequence at the end of the beam-search, we again look at the sum of the log-probabilities and take the one with the value which is nearest to 0 (as a sum of log-probabilities is a sum of negative numbers).

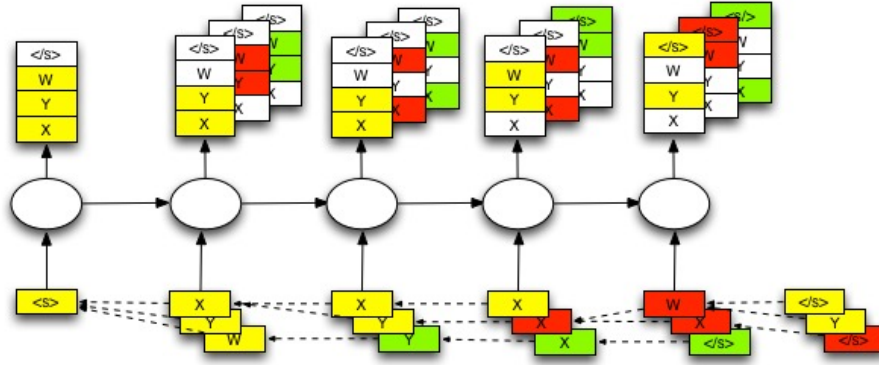


Figure 3.5.: Visualization on how beam search works in the context of seq2seq models.³

3.4. Performance Metrics

In this Section, we are going to introduce the main performance metrics used in this thesis to assess the performance of the trained models.

Cross-Entropy Loss For our model, we are using the *cross-entropy* loss function while training and also for evaluation. It can be used to measure how well an expected probability distribution p is predicted by a trained model, where the predicted distribution is denoted as q . In our case, the probability distribution $p(x)$ is the distribution of the words predicted by the decoder at the time step t and $q(x)$ is the expected distribution for the given sample x . The loss function is defined via the following equation, where X stands for the entire training dataset:

$$H(p, q) = - \sum_{x \in X} p(x) \log_2 q(x) \tag{3.4}$$

In the case of sequence-to-sequence learning, the expected probability is a one-hot encoded vector with a 1 at the entry of the expected word at time step t and 0 everywhere else. This loss function can be used for training a seq2seq model (or any model which predicts probability distributions in a broader sense) and it can then also be used to calculate the perplexity of the model, which we are going to elaborate on below.

Perplexity The perplexity is another metric for measuring how good a language model will predict the sentences in the test dataset and is closely tied to the cross-entropy loss

³<https://github.com/tensorflow/tensorflow/issues/654>

function. The value of the perplexity can be computed by raising the value of the cross-entropy loss function to the power of 2:

$$\text{perplexity}(X) = 2^{H(X)} \quad (3.5)$$

The perplexity tells “how good” a language model is. More formally, it tells us how good the model reproduces the samples seen in the test set. For example, a “dumb” model would predict each word with equal probability, which would be $1/|V|$, where $|V|$ is the vocabulary size. This does not reflect reality, as certain words are much more likely to occur often in language.

Sent2Vec Similarity We were seeking for a third performance metric, since the perplexity is simply 2 raised to the power of $H(X)$. This means, we effectively have one performance metric to use. To fix this problem, we also investigated into using Sent2Vec⁴ [4] for measuring the semantic difference of the sentences generated by the models when testing them after training. We propose the following way to test the models:

1. Allocate a variable for storing the sum of similarities.
2. Repeat the following steps until all samples in the test set are exhausted:
 - a) Create prediction for the given input sentence.
 - b) Embed the generated and expected sentences via Sent2Vec to obtain n -dimensional embedding vectors for both of them.
 - c) Measure the distance by using the cosine similarity between the embeddings in the n -dimensional vector space.
 - d) Add this similarity to the sum of similarities.
3. Divide the sum of all collected similarities by the number of samples. This is the final result which can be used as a metric.

This procedure allows us to further analyze the performance of the models by comparing the semantic similarities between the expected and the generated answers in the n -dimensional vector space, where n is the dimensionality of the sentence embeddings. For example, when the expected output sentence is “I feel good, how about you?” and the answer generated by the model is “I’m fine, you?”, the similarity measurement should return a value close to 1 since we would expect that these sentences are embedded closely to each other due to the semantic similarity of the content. In contrast, when the output of the model is “I really love cupcakes” and the expected sentence is still the same, the similarity measure should become close to 0 to signify that there is a large mismatch between the meaning of the expected and generated sentence.

⁴<https://github.com/epfml/sent2vec>

4. Software System

In this Chapter, we discuss the development of the software system used to conduct the training of the models described later in this thesis. First, we describe the basic requirements for such a system and then go into the “story” behind the development, show where we were stuck and how we solved these problems. Then we describe the environment in which this software system can be used and what kind of external software and hardware was used to run the training on.

4.1. Requirements

We started the development of the system by defining which requirements it has to fulfill in order to be suitable for conducting the trainings of this thesis. The following properties were identified which the developed system should fulfill:

- **Parameterizable:** All trainings should be allowed to run in parameterized manner, which means that all important hyperparameters, training samples and places to store the different results should be parameterizable through a JSON¹ configuration file.
- **Repeatable:** It should be possible to run trainings multiple times without having to put additional efforts into it.
- **Analyzable:** This means that all results from all trainings should be stored in a distinct place for analysis later. This includes the trained models, all metrics collected while training and also the configuration used.
- **Recoverable:** Due to the fact, that we knew that the environment in which the trainings are conducted (see Chapter 4.6 on the hardware) is unstable, we decided that we must design the system in such a way that we should easily be able to recover from the premature termination of a training. This means, that it should be possible to load the most recent model from the previous training and go on from the point where the training was interrupted. This includes loading and saving of the model weights, skipping all of the samples dataset the model was already trained on as well as loading all the metrics already collected in the first run.

¹<http://www.json.org/>

- **Evaluable:** The trained models should be easily evaluable, which means that we should have a way of doing inference without the hassle to load the stored configurations and models by hand. The most comfortable way of doing this is through a small frontend which provides the possibility to communicate with a trained model.

The properties listed above should be fulfilled by the newly developed system in order to enable us to safely conduct trainings without the fear of lose any of the results.

4.2. Development of the System

In this Chapter, we want to give an insight on the development on the software system and shed light on some decision we have made in the process.

Switch from Keras to TensorFlow The first and probably also the largest decision we had to make was, if we would remain with `keras`² as our deep learning framework or if we switch to `TensorFlow`³. We already had developed a software system for conducting experiments with `keras` in the context of our “Projektarbeit” in the autumn semester of 2016, even though this system was used to perform sentiment-analysis with convolutional neural networks [32]. But, from that experience, we also knew that `keras` is a pretty high-level framework with a lot of abstractions to hide the tedious details from the user. This is beneficial for getting up-and-running quickly, but we wanted to get further insight on how these models actually work and how they are implemented, especially with a computational graph, as this seems to be a pretty common technique to build such systems these days [33][34][35].

Due to the urge to get more detailed knowledge, we decided to opt for `TensorFlow` as our framework of choice instead of `keras`, even though the implementation would probably have been easier using the latter.

Beginning Was Hard We had to spend the first few weeks to get some basic knowledge about sequence-to-sequence learning and `TensorFlow` and how its internal graph works. The learning curve was really steep at the beginning, especially when talking about managing the computational graph. In `keras`, everything related to the graph of `theano` is hidden behind abstractions and the user of the framework rarely has to implement a functionality by itself. This is the complete opposite in `TensorFlow`: Most of the time developing the model is spent on the implementation of the graph and even when using specialized `seq2seq` APIs provided, it took a long time to get used to regardless. After our first few experiences with `TensorFlow` in general, we started to concentrate on the model we wanted to implement.

²<https://keras.io/>

³<https://www.tensorflow.org/>

So, we started by using the API in the namespace `tf.contrib.seq2seq` as this seemed to be the one which is under development currently. The implementation itself took around one to two weeks and we started a training with this new model directly after we finished the implementation. After the training had finished, we started to do inference to shockingly notice that the model does not seem to work. The outputs of the model itself were nonsense as they were texts, where it seemed to choose random words and repeat them for a random number of times. We started by analyzing our implementation in detail, but could not find the obvious problem. Driven by the problem, we started to do a research on which of the seq2seq APIs was usable and quickly found out, that the API we thought we were going to use, was unmaintained and we could not seek out to anybody for help regarding our problem.

Ongoing Problems After realizing that the first API we had chosen was not sufficient, we started to search the internet for a solution on which seq2seq API we could use or if we had to implement the whole system by ourself. That was the moment when we noticed, that Google had released a framework⁴ specifically for seq2seq models with TensorFlow. Of course, we were excited to try it out and so we started with conducting trainings using the framework some days later. At first, everything looked fine and the training seemed to be working. We trained a model on the OpenSubtitles corpus and assumed that we could do the first tests by doing inference a few days later. The first problem we faced was that it was not possible to do any validation while training the model. This had to do with a bug⁵ that also troubled us later when doing inference. We nevertheless started training in the hope that this would only be a temporary bug and would not affect us for a prolonged period of time. We were wrong with this assumption, because it took about a month to fix the issue completely, as it was not only caused by a bug in the framework but by a bug in TensorFlow itself. Due to this bug, we were also not able to do inference on a trained model, which worsened our situation even more.

Due to the problems pointed out above, we decided that we would better switch back to our own implementation for the model instead on relying on a framework with serious bugs and flaws which did not seem to be solved in a predictable time frame.

Finally, a Working Model We knew that our own implementation also had bugs, but we were eager to find and fix them. We started by reiterating the implementation of the model on the TensorFlow and noticed, that we had made a colossal mistake which was present in all our implementations before: We *always* initialized the graph with random values when starting a . This is the usual procedure when creating a *new* TensorFlow model, but it is a severe mistake when we are doing that after an already trained model has been loaded. So, we went on to fix this problem. Our second problem was, that we had no clue on how experimental and stable the seq2seq APIs from TensorFlow are. After some more research, we found an example of a working model in a tutorial of the TensorFlow team on how to implement a simple neural machine translation system

⁴<https://github.com/google/seq2seq>

⁵<https://github.com/google/seq2seq/issues/103>

using their framework⁶. However, this implementation relied on a deprecated API in the namespace `tf.contrib.legacy_seq2seq`. We implemented the model and decided, that we somehow had to validate that the model was working as expected. This was the time, when we started to wrap our heads around model validation tests. We came up with two, an overfitting and a copy task, with which we decided to validate our models from now on. It took us two to three days to develop these tests and another day or two for validating the latest model. After these tests were successful, we decided that we would use the latest model using the deprecated API from now on, because in the meantime we have found several other projects using the same API successfully.

However, we still had problems due to the sheer size of the model we wanted to run. To reproduce the results from [1], we tried to make our model as big as possible to come as close to the used 4096 hidden units and 100k vocabulary. This caused another kind of problem, due to the softmax at the end of the decoder. Since the model was so large, the softmax weights matrix alone would have been around 30 to 40 gigabytes in size, which did not fit on any GPU we knew about at the time. To fix this problem at training time, we started to implement sampled softmax as also mentioned in the tutorial for the model we have chosen to use. This caused another chain of problems due to the fact, that the tutorial was written for older version of the TensorFlow framework prior to the one we were using at the time, which meant, that the implementation in our case would be different. After some research and several days of trial-and-error, we managed to fix the problem and start a training using a model with 2048 hidden units and a vocabulary consisting of 50k words. The training run fine and after a few days we had a trained model on our hands and we wanted to start to do inference.

When we started to do inference with our first, successfully trained model, we quickly noticed that our fix for the problem with the size of the softmax weights matrix was only applicable for training time, because at inference time, the model needs the entire softmax weights matrix to make predictions. This led to ourself revisiting the problem and trying to find a solution for it. We found one in the form of an output projection (similar to the one used in [1]), which allowed us to project the last hidden state of the decoder down to 1024 units before feeding it to the softmax layer. This meant, that the softmax weights matrix would be halved in size, which allowed us to run the model on a single GPU without having the out-of-memory problem we have faced before. The implementation of this feature was rather tedious, as we had to adapt the existing seq2seq API inside the TensorFlow framework. But we managed to do it and it allowed us to solve the problem with the too large softmax weights matrix when doing inference on a GPU.

The last part which we implemented for our model was the beam-search. As we did not have much prior experience with a computational graph, this task was much harder to pull off than the others before. We were lucky and found several implementations for

⁶<https://www.tensorflow.org/tutorials/seq2seq>

it scattered throughout the internet^{7,8,9,10}. They helped us a lot in understanding on how beam-search works in the context of a decoder in a seq2seq model. We managed to implement it with the help of an implementation for an older TensorFlow version¹¹. It was quite a struggle to adapt it to the newer TensorFlow version we were using, but we managed to do it.

Summary In summary, our journey to developing a working seq2seq model for our thesis has brought us fun, pain and knowledge, all at the same time. It was a big struggle in the beginning, where we did not find a way to develop a working model and were disappointed by the seq2seq framework we thought would be our savior. The most interesting part of this came after we were able to put our hands on a working model: The implementation of features surrounding the problem of a "too big" model has brought us a lot of joy gave us a lot of insights on how to develop such systems with low-level computational graph operations. The implementation of the beam-search and the down-projection of the hidden state was by far the most interesting part, as we had the possibility to work on internal code from the TensorFlow framework which further deepened our knowledge.

4.3. Model Validation Checks

As described in the Section before, we were struggling with the development of the software system, especially with the construction of the TensorFlow graph itself. We have had a lot of setbacks and hence, were not sure how to validate that our latest model is working without spending several days to weeks on training before noticing that the graph is still broken. For this reason, we have developed two so-called model validation checks, which should ensure that the model is not broken before starting the large, long-running trainings.

The first validation we introduced was a simple copy-page task: The model was trained to copy sequences of integers of different lengths. For this purpose, we have generated a dataset of 10,000 sequences of random length and random content. We have used the integers from 0 to 19 as the vocabulary and the generated sequences were between 1 and 40 symbols long. We then used the implemented model and trained it to output the exact input sequence, for which we used the aforementioned random integer sequences.

The second validation test was a simple overfitting test: The model was fed with tweets as input sequences and the respective response as output sequences. For this purpose, we

⁷https://github.com/google/seq2seq/blob/master/seq2seq/inference/beam_search.py

⁸https://github.com/wchan/tensorflow/blob/master/speech4/models/las_decoder.py

⁹<https://gist.github.com/nikitakit/6ab61a73b86c50ad88d409bac3c3d09f>

¹⁰<https://github.com/stanfordmlgroup/nlc/blob/master/decode.py>

¹¹https://github.com/pbhatia243/Neural_Conversation_Models/blob/master/my_seq2seq.py

have used a dataset of English tweets borrowed from the user "Marsan-Ma" on GitHub¹². We conditioned the model to output the expected output sequence each time we have used a specific input sequence.

Through using these two validation checks, we were able to gain confidence that the implemented model works as expected without using as much time as a full training on any of the conversational datasets would have required.

4.4. Web-UI

We have also developed an elementary GUI for doing inference through a web frontend. This allowed us to interact with the models after they have been trained in a quick and easy way. We have used the python web framework flask¹³ for implementing the backend system and a combination of jQuery¹⁴ JavaScript library and the bootstrap¹⁵ frontend framework.

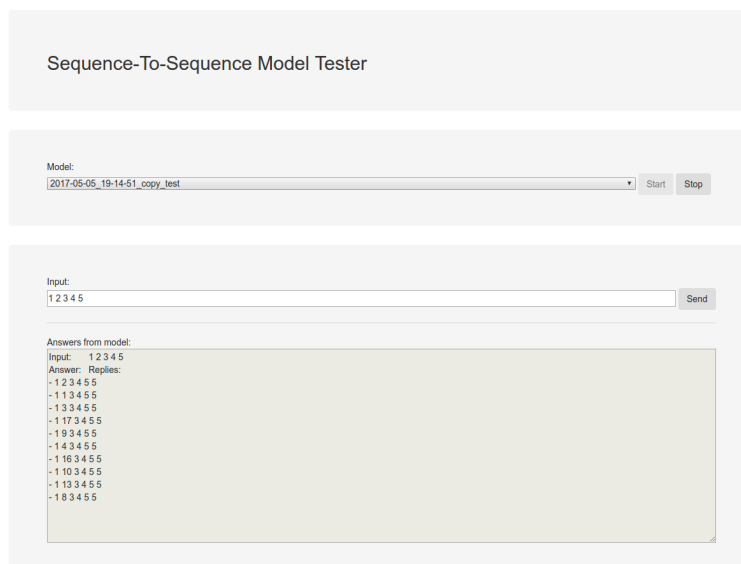


Figure 4.1.: Frontend showing the output when sending the sequence "1 2 3 4 5" to a model trained on the copy task (see Chapter 4.3).

4.5. Scripts

We have written several scripts in order to prepare, conduct and analyze trainings. These scripts cover, amongst other things, the following functionality:

¹²https://github.com/Marsan-Ma/chat_corpus/

¹³<http://flask.pocoo.org/>

¹⁴<https://jquery.com/>

¹⁵<http://getbootstrap.com/>

- Scripts for generating plots of the learning curves and metrics.
- Scripts for analyzing the internal structure of the model.
- Scripts for preprocessing the training dataset and generating required auxiliary files (e.g. vocabularies).
- Scripts for visualizing highly-dimensional data, such as word-embeddings and thought vectors.
- Scripts for visualizing the attention mechanism.

More information about the important scripts and details on how to use them can be found in the appendix B on how to use the software system.

4.6. Hardware

All trainings have been conducted on the GPU-cluster the InIT (Institut für angewandte Informationstechnologie) has provided us for running the trainings related to our thesis. On this server, 8 Nvidia Titan X (Pascal) GPUs are installed and a total of 24 CPUs with 501GB of RAM stood at our disposal. The trainings themselves were run in a virtualized manner by using `docker`¹⁶ with the specialized `nvidia-docker`¹⁷ appliance to ease the integration of the physical hardware into the virtual machine. Most of the trainings have been run using a single GPU of the beforementioned 8, however, one could imagine that multiple of them could be used to speed up the computation and increase the size of the network as this is mainly restricted due to the RAM on a single GPU. However, as this is not easily possible without having to rewrite the whole `TensorFlow` graph, we decided against it and went on with the biggest possible model which fits on a single GPU.

4.7. Operating System & Software Packages

As mentioned before, all trainings have been conducted on the GPU-cluster provided by the InIT. The operating system installed on this server was Ubuntu in the version 16.04. The whole software system has been written in `python`¹⁸, version 3.5.2 using `TensorFlow`¹⁹ in the version 1.0.1. For the GPU integration, we have used the Nvidia GPU driver in conjunction with the `cuda`²⁰ in the version 8.0. In summary, we used the following `python` packages for various different scripts and parts of the system:

¹⁶<https://www.docker.com/>

¹⁷<https://github.com/NVIDIA/nvidia-docker>

¹⁸<https://www.python.org/>

¹⁹<http://tensorflow.org>

²⁰<https://developer.nvidia.com/cuda-toolkit>

Name	Version	Reference
flask	0.12	Website ²¹
gensim	1.0.0	[36]
h5py	2.6.0	Website ²²
networkx	1.11	[37]
nltk	3.2.2	[38]
numpy	1.11.3	[39]
matplotlib	2.0.0	[40]
scipy	0.18.1	[41]
scikit_learn	0.18.1	[42]
TensorFlow	1.0.1	[33]
tqdm	4.11.2	Website ²³

Table 4.1.: Python libraries used in the system.

Additionally, for generating the Sent2Vec embeddings the user needs to have access to the `fastText` [43] binary.

5. Data

In this Chapter, we introduce the two main text corpora used in this thesis. This includes an explanation of their content and how we preprocess it in order to obtain the datasets for training our models. A basic n-gram analysis of the datasets is also prepared in for later usage in the Chapter 7.

5.1. Datasets

We are using two different corpora to build the datasets for our trainings. The first is the current version of the OpenSubtitles corpus [3] that contains subtitles from over 300,000 movies. The second corpus contains all comments posted on the website Reddit¹ from 2006 until 2015. Some general information can be found in Table 5.1.

	Short name	Size [GB]	Lines [Million]	Data format	Source
OpenSubtitles 2016	OpenSubtitles	93	338	XML	[3]
Reddit Submission Corpus	Reddit	885	1'650	JSON	Reddit Comments Corpus ²

Table 5.1.: Origin and some additional information about the raw corpora.

The two corpora differ in several aspects: Since the OpenSubtitles corpus is extracted from subtitles of a large number of movies, it represents a corpus of spoken language. In contrast, the Reddit corpus only consists of comments posted on the website and hence represents written, partially colloquial, language. These two types of language usually differ greatly in how they are used³.

¹<https://www.reddit.com/>

²https://archive.org/details/2015_reddit_comments_corpus

³A good comparison can be found on the website of the University of Westminster: http://www2.wmin.ac.uk/eic/learning-skills/literacy/sp_vs_writ_dif.shtml

5.2. Structure of the Raw Corpora

The following two paragraphs explain how the raw corpora are structured and how the raw utterances are extracted.

OpenSubtitles The OpenSubtitles corpus is structured via XML files, each one containing all utterances from a single movie (see Figure 5.1). Each XML is structured the same way: It starts with the declaration of the `<document>` tag and an `id` attribute signifying which movie the subtitles are from. However, as our research has shown, the value of the `id` attribute cannot be used to decide whether subtitles of a certain movie have already been processed, because there are files with the same `id` but different content. After the `<document>` the actual utterances are listed in `<s>` tags. Each of these `<s>` contains multiple `<w>` tags that contain the actual words uttered in this utterance. Furthermore, one can see that there are also `<time>` tags that contains a timestamp when the utterance started and one when the utterance ended. These timestamps are later used for the time-lag analysis in the Chapter 5.6. There is absolutely no indication on turn-taking, which forces us handle each utterance as an utterance by a different person, even though it could certainly be uttered by the same.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <document id="4955664">
3    <s id="1">
4      <time id="T19S" value="00:01:54,300" />
5      <w id="15.1">Here</w>
6      <w id="15.2">it</w>
7      <w id="15.3">'s</w>
8      <w id="15.4">your</w>
9      <w id="15.5">brain</w>
10     <w id="15.6">versus</w>
11     <w id="15.7">the</w>
12     <w id="15.8">opponent</w>
13     <w id="15.9">'s</w>
14     <w id="15.10">.</w>
15     <time id="T19E" value="00:01:58,250" />
16   </s>
17   ...
18 </document>
```

Figure 5.1.: Example XML entry from the OpenSubtitles corpus.

Reddit The Reddit corpus is structured in huge plain text files for each year, where comments are available. In our case, the comments available are ranging from 2006 to the year 2015. For each year, there is a separate plain text file containing one JSON object per line. Each of these JSON objects (see Figure 5.2) contains a single comment posted

on the Reddit website. There are several different values in each of the JSON objects, however, we only need four of them for our purposes:

- **body**: This attribute contains the actual comment posted on Reddit.
- **name**: This attribute contains a unique ID for each comment posted.
- **link_id**: This attribute contains a unique ID for the thread a comment was posted in.
- **parent_id**: This attribute contains the name id of the comment that this comment answered to.

The shown attributes can be used to create a tree structure per thread, where the `link_id` is used as the root node. The child nodes of the root are the comments where the `parent_id` is equal to the `link_id`. Each of these comments may have several comments answering the original comment; these are matched by comparing the name of the original comment and the `parent_id` of all other comments. The same procedure applied to the comments on the topmost level of the tree is then also applied to their child comments in a recursive manner until a comment does not have any child comments anymore. This process leads to a tree like structure for the comments, which we store in a `shelve`⁴ file, which we then use for building the final dataset (see Chapter 5.3).

⁴<https://docs.python.org/3.4/library/shelve.html>

```

1      {
2          "retrieved_on": 1425820157,
3          "parent_id": "t1_c02s9rv",
4          "distinguished": null,
5          "created_utc": "1199145604",
6          "score_hidden": false,
7          "subreddit": "reddit.com",
8          "score": 4,
9          "author_flair_text": null,
10         "author_flair_css_class": null,
11         "body": "Wow, you're a buzz-kill.",
12         "ups": 4,
13         "id": "c02s9s6",
14         "archived": true,
15         "downs": 0,
16         "edited": false,
17         "subreddit_id": "t5_6",
18         "controversiality": 0,
19         "name": "t1_c02s9s6",
20         "link_id": "t3_648oh",
21         "gilded": 0,
22         "author": "Haven"
23     }

```

Figure 5.2.: Example JSON entry from the Reddit corpus.

5.3. Preprocessing

Steps that are applied to both corpora are mentioned in the “General” paragraph, the rest in the corpora-specific paragraphs.

General The first step is to extract all utterances from both corpora and tokenize them. For this purpose, we use the `word_tokenizer` from the `nltk` library. After extracting all the utterances, we run them through a set of regular expression which ensure that no unwanted characters are present: All characters which are neither alphanumeric (i.e. A-Z, a-z, 0-9) or punctuation (i.e. , , . , ? , !) are removed from the text. The last general step is to convert all characters to lowercase and join the words again with a space in between them (punctuation also counts as words). In Table 5.3 you can find an example of how a text is transformed from its raw form to the preprocessed form.

Raw Utterance	Preprocessed Utterance
Tae Gong Sil was the 'big sun', and you're 'little sun'.	tae gong sil was the big sun , and you re little sun .

Figure 5.3.: Example of an utterance before and after the preprocessing has been applied.

OpenSubtitles After general preprocessing has been applied, we start by building the final dataset from the OpenSubtitles corpus. As we do not have any information about turn-taking or speakers, we have to assume that each utterance is uttered by a different person, even though it might be possible that the same person said both of them. We then write out the preprocessed utterance to a file, one per line. This serves as our dataset and does not need any further processing. As said, we do not have any information about speakers or turn-taking and hence cannot decide whether a conversation has ended or not, this is the reason we do not need a dedicated token for showing that a conversation has ended (as opposed to the Reddit, see next paragraph).

Reddit After the general preprocessing has been applied, we start by building the final dataset from the Reddit corpus and the tree structure explained in Chapter 5.2 (see Figure 5.4). We first filter all comments by subreddit and only include those which were posted in either *movies*, *television* and *films*. Then, we start by reading the tree of the first thread from the stored tree structure. We then go on to read the comments on the first level, in the case of Figure 5.4 this is only A. This comment might have several comments as child nodes; in the example tree these are B, C and D for the comment A. The comments are then combined so that each root comment is combined with all their child comments to form samples. In the case of the exemplary tree, this yields the samples (A, B), (A, C), (A, D), (D, E) and (D, F). This is done for all comments and all levels to obtain all samples for the Reddit dataset. These samples are then written into a plain text dataset, where each line contains one utterance. To separate the conversations, we put a dedicated token (<<<<<END-CONV>>>>>) between them to signify when a conversation has ended.

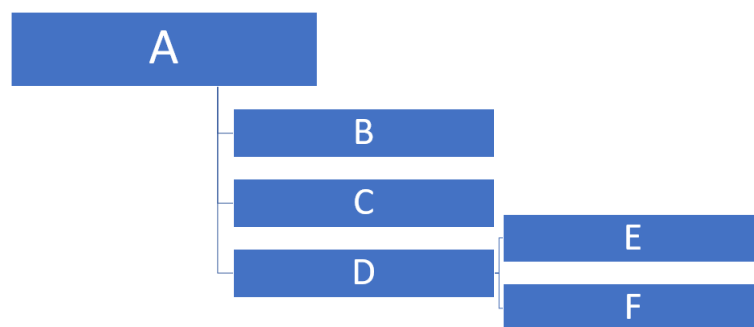


Figure 5.4.: Exemplary tree how the comment tree is structured for a single thread. A stands for the threads root node and B to F are comments.

5.4. Vocabulary and Coverage

After the datasets for both corpora have been generated, we go on with generating dataset specific vocabularies. We do this by iterating the generated dataset and extracting all words that occur there. These collected words are then ranked by their frequency of occurrence and converted into vocabularies containing the n most used words, where n stands for the number of words chosen. We decided that we would use three different vocabulary sizes: 25,000, 50,000 and 100,000. As expected, the larger the vocabularies get, the better the coverage is (see Table 5.2).

	Size [Thousand]	No. of Words [Thousand]	No. of known Words [Thousand]	Share of known Words [%]	No. of unknown Words [Thousand]	Share of unknown Words [%]
OpenSubtitles	25	2362	2096	88.73%	266	11.27%
	50	2362	2116	89.57%	246	10.43%
	100	2362	2127	90.03%	236	9.97%
Reddit	25	1717	1683	98.00%	34	2.00%
	50	1717	1699	98.98%	17	1.02%
	100	1717	1707	99.42%	10	0.58%

Table 5.2.: Word coverage of differently sized vocabularies extracted from the generated datasets.

Figure 5.5 shows what percentage of the words in an utterance is missing using the specified vocabulary. It shows that the coverage of the Reddit vocabularies is much bigger than the OpenSubtitles.

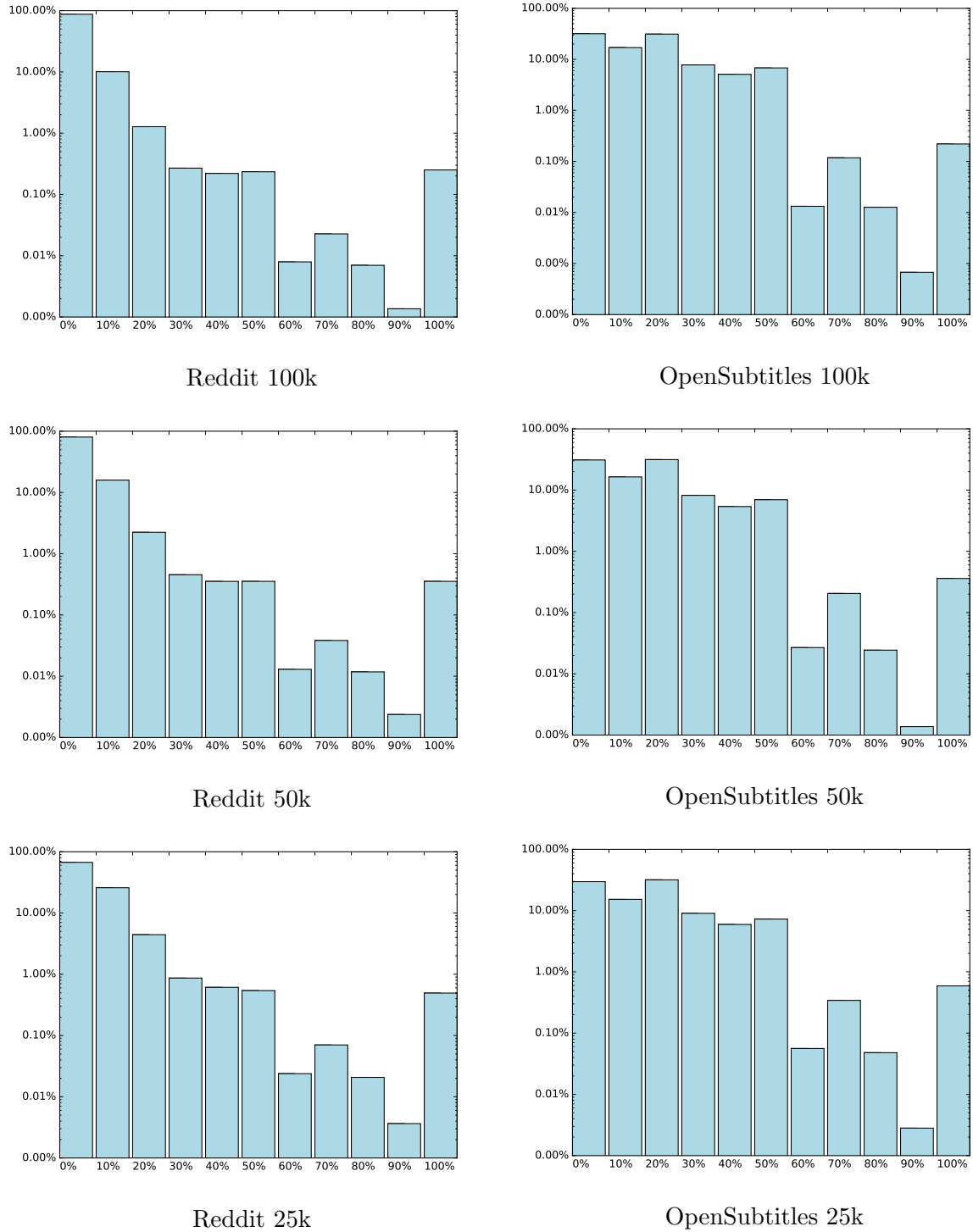


Figure 5.5.: Semi-logarithmic histogram showing the percentage of words missing per utterance for specific vocabulary sizes.

Another thing worth mentioning are the special tokens in each of these vocabularies. There are four of them in total: `<unknown>`, `<pad>`, `<eos>` and `<g0>`. The first one is used in case there is word in a sample which is not covered by the vocabulary. In this case, instead of the missing word, the `<unknown>` token is fed to the model. The `<eos>`

token is appended at the end of each sentence and the `<pad>` token is used to pad the sequences to the required length. The `<go>` token is used to feed it as the first token to the decoder.

5.5. Splitting the Datasets

The generated datasets have to be split in order to obtain a train, validation, and test datasets. The Table 5.3 shows proportions in which the datasets were split. We split in a way, that we always process two utterances at one to ensure that there is no overlap between the training and the other datasets. The splitting is also done randomly, to ensure that we do not run into problems due to the fact that the utterances may be sorted in any unknown way.

	Dataset	Share of Dataset [%]	Size [MB]	No. of Lines [Thousand]
OpenSubtitles	Train	97%	9'393	321'643
	Valid	1%	97	3'315
	Test	2%	194	6'631
Reddit	Train	97%	8'455	75'297
	Valid	2%	185	1'552
	Test	1%	92	776

Table 5.3.: Proportion in which the datasets were split to obtain a train, validation and test dataset.

5.6. Time-Lag Analysis OpenSubtitles

While searching through the OpenSubtitles dataset, we discovered that there are several pairs of utterances which don't make a lot of sense (e.g. "I love cupcake", "Hello, how are you?") under the viewpoint of conversational modeling. We quickly realized that this useless pairs of utterances occur because of the way we preprocess the raw corpus: In order to get a sample, we take the first utterance and combine it with the utterance on the next line in the preprocessed corpus. The problem occurs when the timestamps signify a large time difference between the first and second utterance. This is understandable, as all utterances are just saved in the corpus one after the other without taking timestamps (see OpenSubtitles paragraph in Chapter 5.2) into account. In order to investigate if this problem is urgent, we did an analysis to see if what proportion of utterances have a time-lag higher than a certain timespan. The results of this investigation are summarized in Figure 5.6.

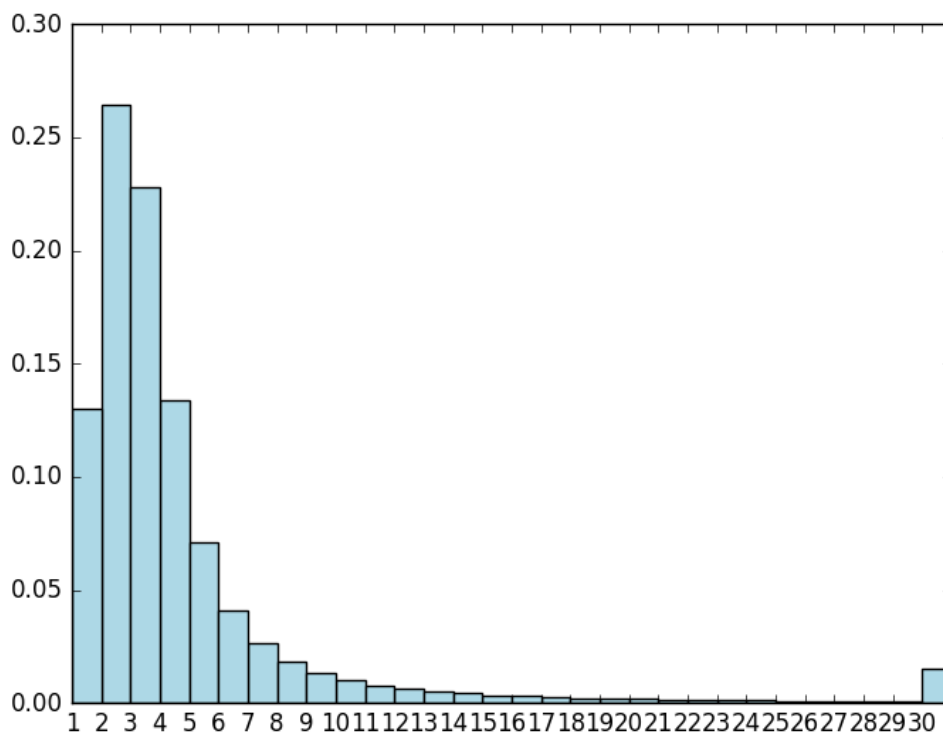


Figure 5.6.: Relative distribution (discrete) of the time-lag between two utterances in the raw OpenSubtitles corpus. Most of the utterances lie in the range from 1 to 5 seconds: 13.0% within 1 second, 26.4% within 2 seconds, 22.8% within 3 seconds, 13.4% within 4 seconds and 7.0% within 5 seconds.

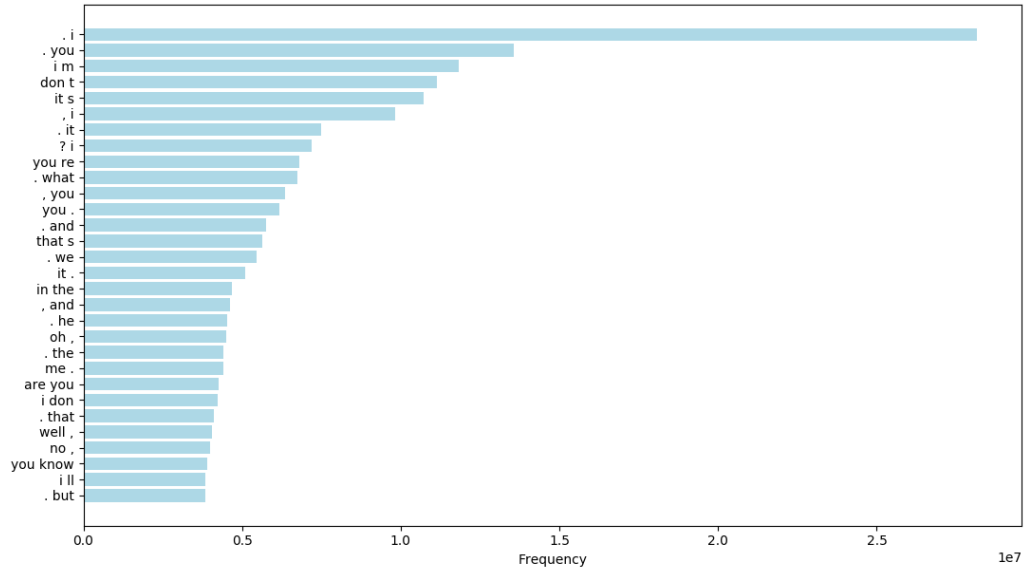
As one can see, over 80% of the utterances occur within one to five seconds of one another. The spike at the end of the graph occurs because there might be changes in the scenes of the movies, which can lead to two consecutive utterances in the corpus being uttered after a larger gap. The other reason is, that each movie is stored in a single file and hence, there might be utterances which are combined, but actually belong to two different movies.

As this analysis has shown, most utterances lie so close to one another that this will most probably not be a problem when training our model with them.

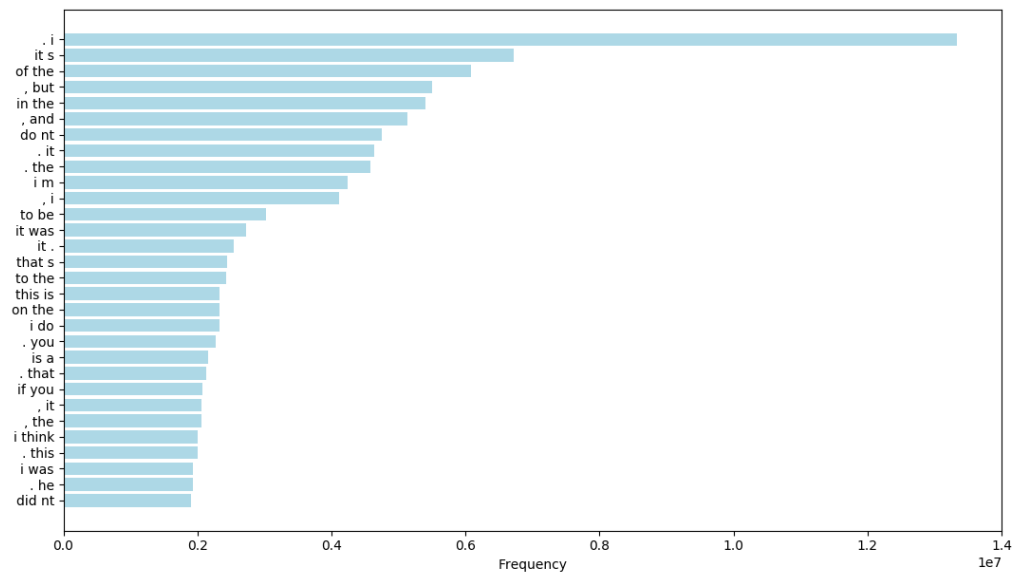
5.7. N-Gram Analysis

We also do an n-gram analysis of the used datasets. For this purpose, we generated uni- and bi-grams for both the OpenSubtitles and Reddit dataset. We do this so that we can compare the n-grams of the outputs produced by our models after they have been trained with the n-grams obtained when analyzing the dataset. This enables us to find

correlations between the n-grams produced by this analysis and n-grams extracted from the outputs of the trained models. The result of analyzing the preprocessed datasets is visualized in the Figure 5.7.



OpenSubtitles



Reddit

Figure 5.7.: Occurrence frequencies of the 30 most used bi-grams in the OpenSubtitles (left) and Reddit (right) datasets.

6. Methods

In this Chapter, we elaborate on how we performed the experiments, which hyperparameters were used, and how we evaluated the results of the trained models.

6.1. Architecture of the Sequence-To-Sequence Model

In the following paragraphs, we describe the architecture of the model used to conduct our experiments.

Sequence-To-Sequence In general, we are using the architecture of seq2seq models described in Chapter 3.3 with LSTM cells. We restrict ourselves to use only one LSTM cell for the encoder, and another cell for the decoder. This is because we wanted our cells to be as large as possible to come as close to the size of the cells used in the paper by Vinyals and Le [1]. However, they used two really large cells with each having a hidden state size of 4096 hidden units and a vocabulary consisting of 100,000 words. In the paper, they had to train their models on a large number of CPUs, because such a huge network does not fit in the memory of any currently available GPU. Since we want to train our models on a single GPU (see Chapter 4.2), we had to shrink the size of our model to the largest size possible so that it still fits within the 12GB of memory that our GPUs have (see Chapter 4.6). The exact size of the model used in this thesis is described in “Hyperparameters” below.

Down-Projection of Hidden State As we have seen above, very large RNN cells are problematic. So, we implemented what is known as a *down-projection* at the end of the decoder cell. This is done similar to the down-projection used in [1], with the main difference lying in the motivation why we implemented it. In the paper, they state that they used the down-projection technique to speed up the training due to the large weights-matrix in the softmax layer at the end. In our case, the down-projection was not just for speeding up the training, but mainly to allow us to use larger cells than we would otherwise be able to. The implementation of this feature allowed us to grow our model by a factor of two, from a hidden state size of 1024 to 2048 units, without having to sacrifice the size of the vocabulary (see Chapter 6.3).

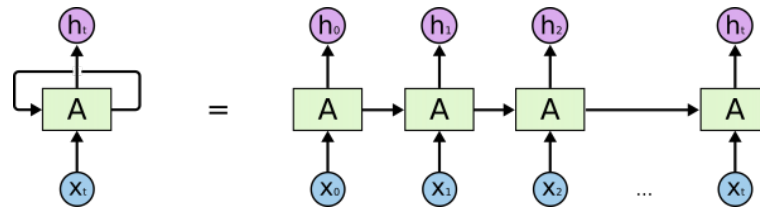


Figure 6.1.: Image for illustrating unrolling an RNN over a fixed size of time steps.²

Sampled Softmax To speed up the training of the large softmax layer at the end (consisting of 50,000 entries), we use a *sampled softmax* as described in [44] which is only applied while training the models. The idea is that, instead of using the full vocabulary for the softmax at each time step, we only use a small random sample (which must, obviously contain the target word). This speeds up training dramatically due to the fact that not the entire softmax layer has to be used and hence, not all derivatives have to be computed. On inference time, when we generate predictions, we have to use the full softmax layer again to generate the predictions.

Static Unrolling of RNN Because we are working with a deprecated TensorFlow API (see Chapter 4.2), we have to use static unrolling for our model. Static unrolling works by defining a fixed size of time steps for the encoder and decoder, and then unrolling the encoder and decoder cell for this number of time steps *before* we are actually computing anything using it. This actually “removes” the recurrence from our model and transforms it into kind-of “feed forward” model with the major difference being that the weights are shared between the layers (as each layer represents the same cell). With the latest TensorFlow API¹, it is possible to create the graph for the encoder dynamically at runtime based on the length of the input. The decoder, in the dynamic case, then runs for as many time steps as necessary to either reach an EOS token or the maximum number of time steps allowed to decode.

This implementation forced us to define a maximum number of time steps to use for the encoder and decoder beforehand, which we have set to 30 for both the encoder and decoder. This length is also mainly restricted due to the fact that the longer the time span becomes, the larger the memory consumption on the GPU becomes. We tried several different sizes ranging from 10 to 100 time steps and evaluated that 30 would be the largest possible size without being required to shrink any other parts of the model.

Reversing the Input Sequence As Sutskever et al. [9] have noticed in their paper, it was beneficial to reverse the input sequence before feeding it to the model. They assume, that this helps because the minimal time lag [20] is greatly reduced due to the fact that the first few words in the source language are much closer to the first few words in the target language. This explanation makes sense in the context of machine translations, but not necessarily in the context of a conversational dialog systems. Nev-

¹https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn

²<http://colah.github.io/posts/2015-08-Understanding-LSTMs/>

ertheless, we want to try to use it and see if it makes any difference to our models. We decided that the Reddit model should be the one that is trained with reversed input sequences.

6.2. OpenSubtitles and Reddit Models

As already described in Chapter 5, we are going to train two distinct models that each use a separate training, validation and test dataset. The first is trained on the OpenSubtitles dataset and the second on the Reddit dataset (see Chapter 5). Both of them are trained in the same way for the same time period, except for the difference regarding how the input sequences are fed to the model.

6.3. Hyperparameters

Model The hyperparameters used for training the models can be seen in Table 6.1. Both of our models use the same set of hyperparameters, the only difference being that the Reddit model is fed with reversed input sequences whereas the OpenSubtitles model is not.

Name	Value
Number of encoder cells	1
Number of decoder cells	1
Max. number of time steps in encoder	30
Max. number of time steps in decoder	30
Hidden state size	2048
Projected hidden state size	1024
Number of sampled words for softmax	512
Size of the softmax layer	50'000
Batch size	64
Max. Gradient Norm for Clipping	10.0

Table 6.1.: Hyperparameters used for our seq2seq models.

Optimizer As the optimizer, we use *AdaGrad* [45] as in [1] with the learning rate set to 0.01. We also use gradient clipping [16] and set the maximum allowed gradient norm value to be 10.

6.4. Training

We train the Reddit and OpenSubtitles models on the hardware described in Chapter 4.6 with the software packages from Chapter 4.7. As these models take a long time to achieve progress (in [1] the authors trained their model for several weeks), we also have to train ours for a prolonged time period. To show the development of the model throughout this time, we take several snapshots, approximately every third day or about after every 500,000 batches. Each of these snapshots has a distinct name indicating at which point in the training it was created. The names are going to follow the naming convention xM, where x stands for the decimal number signifying the number of batches processed in millions (e.g. 0.5M stands for 500,000 and 1.5M stands for 1,500,000 batches processed). The taken snapshots are going to be used in the analysis we do in the next Chapter. They enable us not only to do an analysis of the final results, but rather show the evolution of the models during the training process. The two models were trained over a time period of 20 days, each trained on roughly 3 million batches of 64 samples each.

As the two training datasets are not equal in size, using the same time period for the training of both models induces that they see different shares of their respective training datasets. In the case of the OpenSubtitles model, the case is that it can only process 192 million samples, which equals to about 59% percent of the whole OpenSubtitles training dataset. In the case of the Reddit model, the two week training allows doing two and a half epochs over the entire dataset. This allows us to analyze the difference between using the approaches of using a single big dataset, which we are not able to completely process, against a smaller corpus that we can iterate more than twice in the same time.

6.5. Evaluation

We use two different kinds of evaluation: Metric-based and human-based. The first one is a quantitative analysis of the results and consists of analyzing the training process and the final results under the metrics defined in Chapter 3.4. We used the test datasets generated as described in Chapter 5.5. Due to the sheer size of the datasets, we only use the top 250,000 samples from the test datasets for our evaluation. We are also going to do an n-gram analysis, as already mentioned in Chapter 5.7, where we analyze the language model of the resulting models with regard to the language used in the datasets. This also includes an analysis of the semantics of the outputs generated by the trained models. The human evaluation is done by judging how “meaningful” the texts generated by the resulting model are and we compare them with results found in the “Neural Conversational Model” [1] paper and the CleverBot³, which is available online for free.

We also considered using the BLEU and ROGUE metrics for evaluating our models. After some research, we have found that this metrics are primarily used in the context of machine

³<http://www.cleverbot.com/>

translations and work by comparing the resulting texts to several reference translations. In our opinion, this does not make a lot of sense in the context of conversational models since the answers to a specific input can have a wide variety of possible forms that are all meaningful and fitting but do not necessarily need to be alike the reference response. This thought is supported by results that Nguyen et al. [46] have found in their paper, where they state that they used the BLEU and ROGUE metrics for evaluating their chatbot and determined that these metrics are not fit to do such an evaluation because of the aforementioned reasons. Instead of the mentioned similarity metrics, we are going to use Sent2Vec [4], which allows us to do similarity measures between the expected and generated outputs of the model using a highly dimensional vector space where the sequences can be embedded. This can be done by using the pretrained models found in the GitHub repository⁴, as it is possible to embed new sequences without the requirement of retraining the models. The exact algorithm for the evaluation using Sent2Vec can be found in Chapter 3.4.

⁴<https://github.com/epfml/sent2vec>

7. Results and Discussion

We are going to analyze the resulting models after training them as specified in Chapter 6.4.

In the first part, we are going to analyze how the training went and review the results of the metric-based evaluation (see Chapter 6.5). Because we identified some problems with this quantitative analysis, we then show with examples that progress was indeed achieved throughout the training and explore why the results of the evaluation are as modest as they are. This includes an analysis of the language model to determine why the models often respond with generic outputs.

The next part is dedicated to the comparison of our models with the results from the paper of Vinyals and Le [1] and with the CleverBot chatbot.

The subjects of the last part are the analyzes related beam-search, the soft-attention mechanism and the clustering of thought vectors.

7.1. Was the Training Successful?

First, we start by analyzing the evolution of the models with regard to the available performance metrics during the time period of the training. Below, in Figures 7.1 and 7.2, the development of the cross-entropy loss and perplexity values on the training datasets for the two different models is depicted.

OpenSubtitles It is eye-catching when comparing the two models that the OpenSubtitles apparently has much more variance in its performance in comparison to the Reddit model. This is most probably caused by the fact that the OpenSubtitles dataset is much more noisy than the Reddit dataset, as also noticed by others [1]. This is related to the missing information about turn taking, which means it is certainly possible that consecutive utterances in the dataset may be uttered by the same person even though we treat it if they were uttered by two different persons. Also, there is the issue with the time lags between utterances as analyzed in Chapter 5.6. In contrast, with the Reddit dataset we always know who uttered a comment and hence can build a dataset which ensures that the dialogs make sense from a structural perspective.

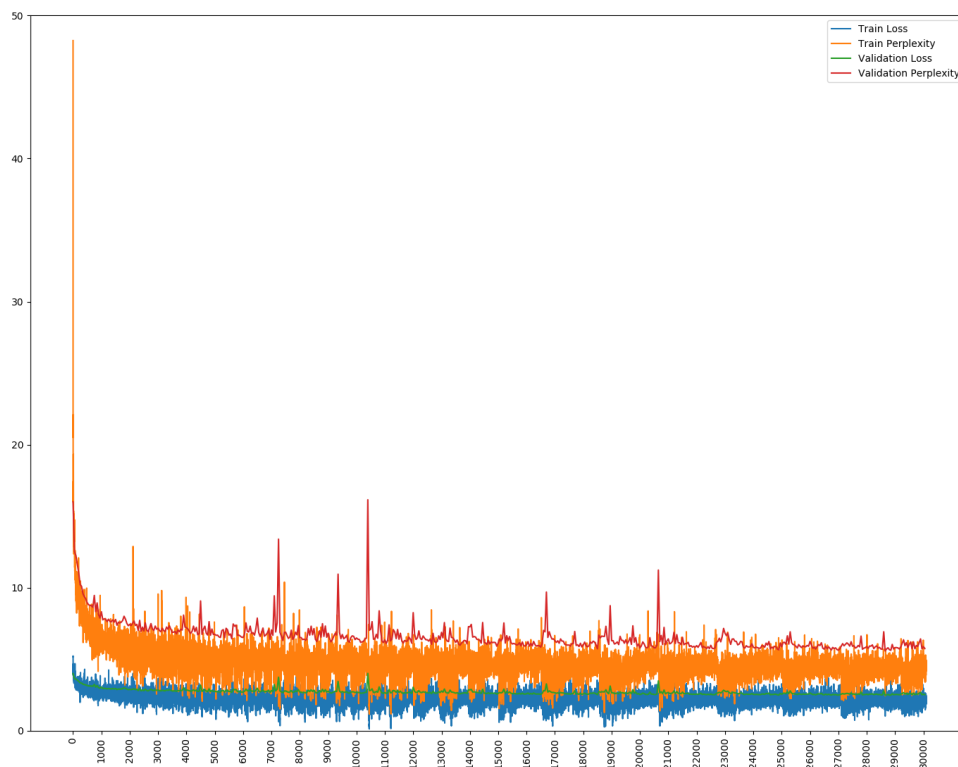


Figure 7.1.: Development of the loss and perplexity values on the training and validation datasets throughout the training of the OpenSubtitles model. One tick on the x-axis is equal to 100 batches processed.

Reddit The learning process of the Reddit model looks appropriate, but it also has a peculiarity, namely the dips in the training loss and perplexity. These dips occur about every 300,000 to 400,000 batches. They are also present in the development of the validation loss and perplexity, but are not as apparent as in the metrics on the training dataset. We currently cannot explain this behavior. We assume that this peculiarities are caused by the structure of the training dataset. However, the variance is much smaller than with the OpenSubtitles model, which strengthens our argument that well-structured datasets are indeed favorable when training such systems as helps to avoid confusion due to perplexing samples.

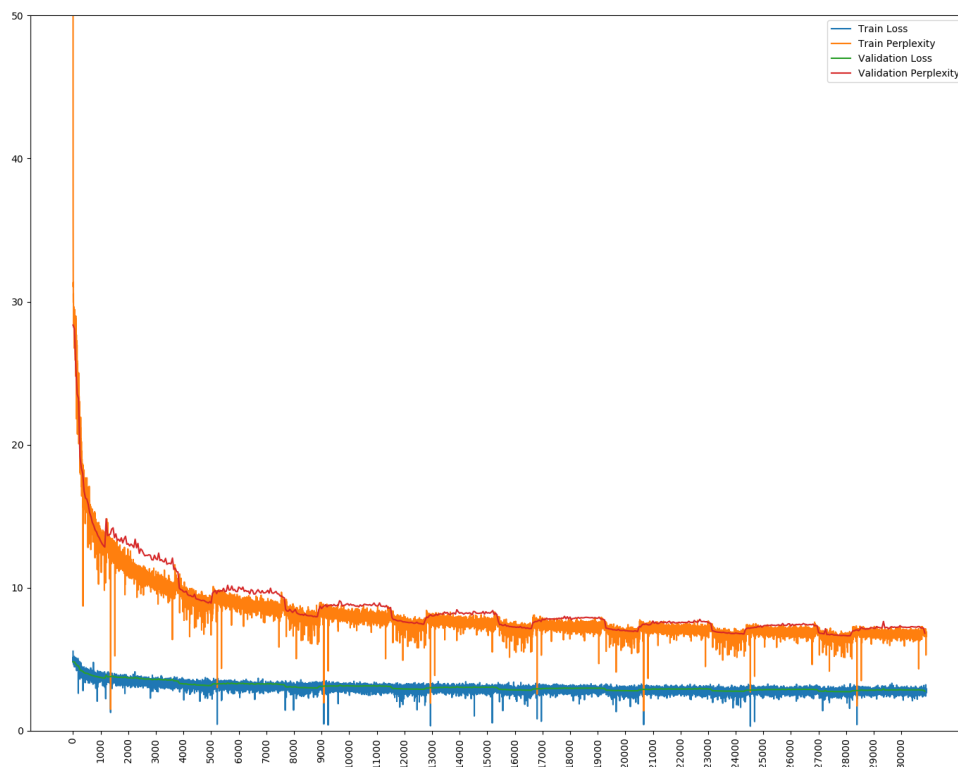


Figure 7.2.: Development of the loss and perplexity on the training and validation datasets throughout the training of the Reddit model. One tick on the x-axis is equal to 100 batches processed.

The Training Seems Successful From the appearance of the plots, it looks like the training went fine for both models, as both of them exhibit decreasing loss and perplexity values. We see differences in how the models have evolved over the time span of the training, but we cannot derive any conclusion at the current time. After we have analyzed the training process, we are now focusing on the performance of the models on the test datasets.

7.2. Performance on Test Datasets

After we have seen that the training process looks fine, we are going to assess the performance of these models on our test datasets. Here we use the same metrics as during the training, namely the cross-entropy loss and perplexity. We evaluate each model on the respective test dataset for each of the six snapshots, we have created during training (see Chapter 6.4).

Surprising Results The results on the test datasets are quite the opposite of the results of the training process, the performance for both of the models is worsening over time. The results of the OpenSubtitles model (see Table 7.1) vary across the different snapshots, with the best result having a perplexity of 71.08 and a loss of 6.15 and derived from the evaluation of the first snapshot. Also, the best result of the Reddit model (see Table 7.2) is achieved on the first snapshot with a perplexity of 169.14, with all other snapshots having a worse perplexity.

Snapshot	Test Loss	Test Perplexity
0.5M	6.1513	71.0779
1.0M	6.5314	92.5000
1.5M	7.3942	168.2207
2.0M	6.2134	74.2035
2.5M	6.3627	82.2949
3.0M	6.4647	88.3205

Table 7.1.: Loss and perplexity values for each snapshot of the OpenSubtitles model when evaluating it with the test dataset.

Snapshot	Test Loss	Test Perplexity
0.5M	7.4021	169.1432
1.0M	7.5477	187.1090
1.5M	7.5794	191.2557
2.0M	7.6288	197.9190
2.5M	7.6661	203.1056
3.0M	7.7885	221.0843

Table 7.2.: Loss and perplexity values for each snapshot of the Reddit model when evaluating it with the test dataset.

This result contradicts with our expectation. Instead increasing loss values, we would have expected it to decrease in the same way as it did on the training and validation datasets. We assume, that this is related to the cross-entropy loss and hence the perplexity being not the best fit metrics to evaluate such models, especially in a conversational context where the variety of correct answers can be extensive. For this reason, we propose a third performance metric, namely the usage of Sent2Vec [4] embeddings, to measure the similarity between the expected and generated responses. Before we perform this analysis, we want to take a look at different samples from both models to show that they indeed improved over time, even though the results of the test metrics tell a different story.

Is Progress Achieved? As the results of evaluating the models on the test dataset has shown, it seems that the models achieve a meaningful progress during the training.

However, in the performance test, it appears like the models have become poorer over the course of the training. For this reason, we decided that we would like to evaluate the model manually and find out if the generated responses improve when using the six snapshots of the models. We have chosen four different input utterances, two for each model, and generated responses for each of the snapshots. The results can be seen in the Table 7.4 for the OpenSubtitles model and Table 7.3 for the Reddit model.

Input: Do you like Joffrey from game of thrones?

0.5M: i m not sure if you re a fan of the movie ,
but i m not sure if you re a fan of the movie .

1.0M: i m not a fan of the game , but i m not sure if it s a
game or not .

1.5M: i do nt know what i m talking about .

2.0M: i do nt

2.5M: i do nt know what i m talking about .

3.0M: i do nt know , i just do nt like him .

Input: Which movie can you recommend me?

0.5M: the

1.0M: the movie .

1.5M: the one with the

2.0M: the dark knight rises

2.5M: the prestige

3.0M: the movie

Table 7.3.: Two exemplary dialogs with the Reddit model, one response per snapshot.

Input: Are you a man or a woman?

0.5M: no , no , no

1.0M: no

1.5M: i m not a man

2.0M: i m not a woman

2.5M: i m a man

3.0M: i m not a woman

Input: Why should it not be a good idea to improve you?

0.5M: no

1.0M: i don t know

1.5M: because i love you

2.0M: because i m a good man

2.5M: i m just trying to make a good decision

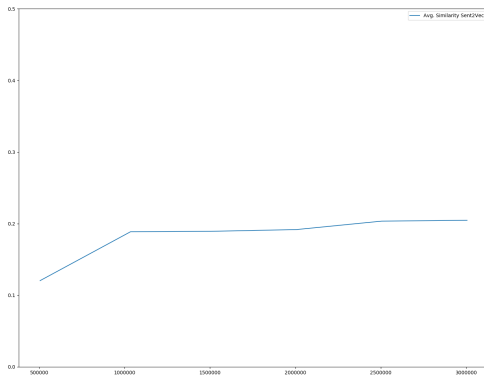
3.0M: i m not a good idea

Table 7.4.: Two exemplary dialogs with the OpenSubtitles model, one response per snapshot.

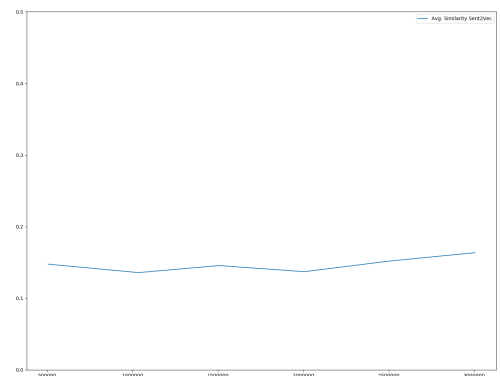
As seen in the examples above, there has indeed been an improvement in the answers from 0.5M and 1.0M to the better trained models, what stands in contradiction to the development of the performance of the models on the test datasets. We suspect from this result, again, that the cross-entropy loss and perplexity are not fit to assess if the responses are meaningful. As we have already described in Chapter 3.4, we were aware of this potential issue; that is why we decided to use an additional metric, namely Sent2Vec embeddings, which we are going to use in the next Section.

Sent2Vec Analysis As described in Chapter 3.4, we leverage Sent2Vec embeddings for measuring the semantic similarity between the generated and expected responses on the test datasets. For this purpose, we have used the pretrained embeddings available on the GitHub page of the project¹. We decided, that we use both the *Twitter* and *Wikipedia* embeddings for the assessment. The results can be seen in the Figures 7.3 and 7.4. It is obvious that both the Reddit and OpenSubtitles models perform better on the pretrained *Wikipedia* embeddings in comparison to the pretrained *Twitter* embeddings. This is probably related to the “compressed” language used when writing tweets (e.g. “w/o” instead of “without”). In conclusion, the results of this analysis are dependent on the Sent2Vec embeddings, as expected. However, both results are poor, with the results from the Reddit model being about twice as good as results from the OpenSubtitles model (see Tables 7.5 and 7.6). The OpenSubtitles model starts with an average similarity of 0.168 for the first snapshot and rises up to 0.205 for the last snapshot. The Reddit model starts with an average value of 0.337 for the first snapshot and increases to 0.360 for the last snapshot. This means that the responses of the Reddit model match the expected responses much better from a semantic perspective as the responses of the OpenSubtitles model do. Our assumption is that this is, at least partially, related to the difference between written and spoken language found both in our datasets. It might be harder to learn from spoken language in our dataset as the conversations are extracted from movies, which leads to several subtle sources of information like pronunciation and body language being lost. Also, we are missing information about turn-taking, which makes the OpenSubtitles dataset more noisy. In summary, the results of both models are unsatisfactory, as the maximum achievable similarity is 1.0.

¹<https://github.com/epfml/sent2vec>



Wikipedia

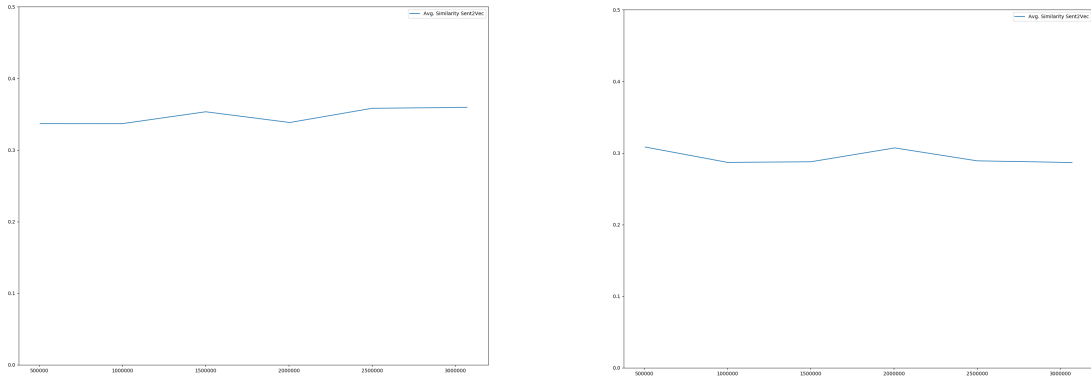


Twitter

Figure 7.3.: Results of the evaluation with Sent2Vec on the outputs of the OpenSubtitles model when using the test dataset. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity for each snapshot.

Snapshot	Avg. Similarity (<i>Wikipedia</i>)	Avg. Similarity (<i>Twitter</i>)
0.5M	0.16749	0.13827
1.0M	0.19111	0.13811
1.5M	0.19418	0.14831
2.0M	0.19176	0.13840
2.5M	0.20118	0.15258
3.0M	0.20452	0.16285

Table 7.5.: The average similarities when using the Sent2Vec metric on the expected and generated responses from the OpenSubtitles model per snapshot.



Wikipedia

Twitter

Figure 7.4.: Results of the evaluation with Sent2Vec on the outputs of the Reddit model when using the test dataset. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity for each snapshot.

Snapshot	Avg. Similarity (<i>Wikipedia</i>)	Avg. Similarity (<i>Twitter</i>)
0.5M	0.33691	0.30837
1.0M	0.33689	0.28694
1.5M	0.35340	0.28777
2.0M	0.33843	0.30713
2.5M	0.35828	0.28908
3.0M	0.35956	0.28676

Table 7.6.: The average similarities when using the Sent2Vec metric on the expected and generated responses from the Reddit model per snapshot.

Generic Responses Are a Problem A potential reason for the poor results when using the Sent2Vec metric could be based on our observation that both models often generate generic responses (e.g. “i don t know”, “i m not sure what you re saying”). Our first idea to analyze this is to see what kind of sentences the models produce with the inputs of the test datasets. We did an analysis on the generated responses and quickly noticed that there are a few sentences, which are often predicted by the models (see Tables 7.7 and 7.8). As one can see in the column *Share of Total Count*, these generic responses take up a great share of all the generated responses. The top 10 responses from the OpenSubtitles model make up 45.77% of all outputs, where the same for the Reddit only sums up to 33.02% of all outputs.

Sentence	Occurrence Frequency	Share of Total Count
i m not gon na let you go	41853	16.74%
i m not sure i can trust you	21263	8.51%
i m not gon na say anything	9163	3.67%
i m not gon na let that happen	7426	2.97%
i m sorry	7235	2.89%
you re not gon na believe this	7068	2.83%
you re not gon na believe me	6878	2.75%
i m not gon na hurt you	4829	1.93%
i m not a fan	4468	1.79%
i m not sure	4215	1.69%
Summed up	114408	45.77%
Total Sentences Count	249984	100.00%

Table 7.7.: Top 10 most generated responses with respective occurrence frequencies when using the last OpenSubtitles snapshot and the test dataset.

Sentence	Occurrence Frequency	Share of Total Count
i m not sure if i m being sarcastic or not .	17486	7.00%
i think it s a bit of a stretch .	13058	5.22%
i m not sure if you re being sarcastic or not .	11647	4.66%
i m not sure if i m a <unknown> or not .	8307	3.32%
i m not sure if you re joking or not .	7932	3.17%
i was thinking the same thing .	7579	3.03%
<unknown>	6210	2.48%
i m not sure if i m going to watch this or not .	4257	1.70%
i m not sure if i m a fan of the show , but i m pretty sure that s a <unknown> .	3232	1.29%
i m not sure if i m going to watch it or not .	3079	1.23%
Summed up	82797	33.02%
Total Amount Sentences	249984	100.00%

Table 7.8.: Top 10 most generated sentences with respective occurrence frequencies when using the last Reddit snapshot and the test dataset. The meaning of the <unknown> token is described in 5.4.

Does Filtering of Generic Responses Help? As seen in the Tables above, there are certain responses which are generated a lot of time and are pretty generic and meaningless in most contexts. Because of that, we think it would be a good idea to evaluate the models under the Sent2Vec metric one more time, but this time with the top n most generated responses filtered out. We do this expecting that the generic sentences are the cause of the small average similarities. The results of the analysis with the top n most generated responses filtered out is shown in Table 7.9 and 7.10. For this analysis, we only use the *Wikipedia* embeddings as it has shown a better performance for both of our models before.

Snapshot	$n = 1$	$n = 5$	$n = 10$
0.5M	0.16679	0.16804	0.16854
1.0M	0.19329	0.19394	0.19575
1.5M	0.19491	0.19519	0.19539
2.0M	0.19215	0.19192	0.19284
2.5M	0.20102	0.20127	0.20182
3.0M	0.20431	0.20547	0.20568

Table 7.9.: The average similarities when applying the Sent2Vec metric on the expected and generated responses on the test dataset when filtering out the top n most generated responses for the OpenSubtitles model.

Snapshot	$n = 1$	$n = 5$	$n = 10$
0.5M	0.33772	0.34101	0.34589
1.0M	0.34225	0.34238	0.34295
1.5M	0.35383	0.35605	0.35564
2.0M	0.34008	0.34009	0.34198
2.5M	0.35937	0.36142	0.36175
3.0M	0.36043	0.35950	0.36313

Table 7.10.: The average similarities when applying the Sent2Vec metric on the expected and generated responses on the test dataset when filtering out the top n most generated responses for the Reddit model.

As seen above, the filtering of the most used responses does not help significantly when it comes to the Sent2Vec evaluation. We currently cannot say if this problem is just apparent in our specific use-case or inherent to the metric itself. To come to a definitive conclusion, we would have to investigate further by using this metric for evaluating other models.

Mixed Feelings about Performance Metrics As seen in this Chapter, the performance metrics we use to evaluate the models tell us an indifferent story about the resulting models. On one hand, we see that the training performed acceptable and the learning process ran as expected. However, as we subsequently test these trained models against the test datasets with the different metrics, it looks like the performance decreased over the duration of the training. Our subjective opinion could not confirm this after we were “talking” to both models for an extensive period of time. We think the biggest problem for the evaluation is the large portion of generic responses both models seem to generate. To identify the cause of this generic responses, we will now analyze the development of the language model while training, trying to establish a connection between the language model in the datasets and the ones produced by the models.

7.3. Development of Language Model

In this Chapter, we are investigating into the language models produced by the training and try to identify a reason for the generic responses.

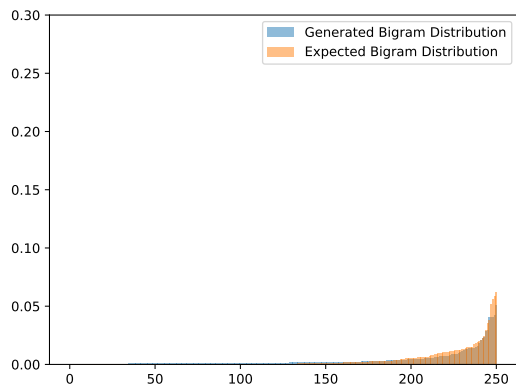
7.3.1. Uni- and Bi-Gram Distributions over Time

Initially, we are comparing the uni- and bi-gram distributions of the training datasets with the distributions produced when evaluating the models with the test datasets. For this purpose, we have generated uni-gram and bi-gram statistics using `nltk` on the training datasets and outputs generated by the models.

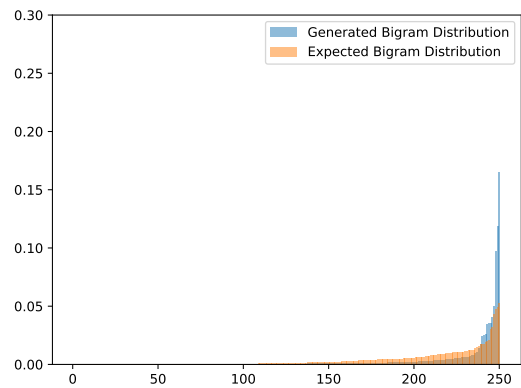
OpenSubtitles The OpenSubtitles has the most evenly spread distribution (see Figure 7.5) when using the first snapshot (i.e. 0.5M). We observe a pattern when we look at all distributions of the OpenSubtitles model. At the beginning, it starts with a rather evenly spread distribution of uni- and bi-grams. As the training proceeds, these distributions shift to a more right-leaned when looking at the distributions for the snapshot 1.0M. The distributions then return to be more evenly distributed again. This pattern repeats until the last snapshot, with one being evenly distributed whereas the next is then again more right-leaned.

We cannot provide a complete explanation for this behavior, but we assume that this is related to the rather noisy dataset we are using for the OpenSubtitles model. This probably leads to the model being confused, rejecting what it has already learned and trying to fit to the newly provided samples. Because of the fact that we cannot fully explain this behavior, we are going to analyze the development of the diversity of the uni-, bi-gram and sentences over the course of the training.

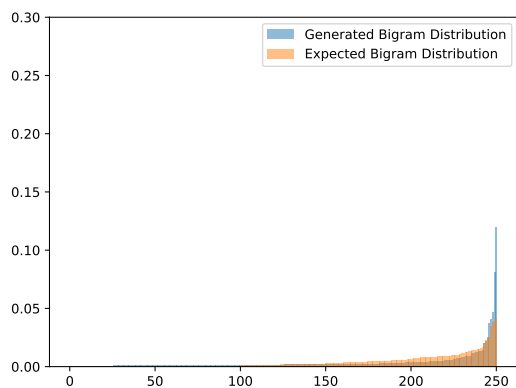
The behavior of the uni-gram distributions develops in a similar way, therefore we documented these visualizations in the Appendix C.



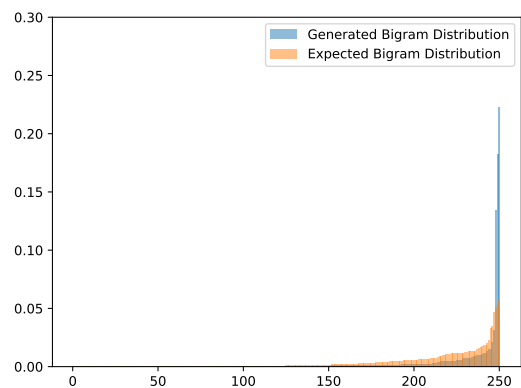
Snapshot 0.5M



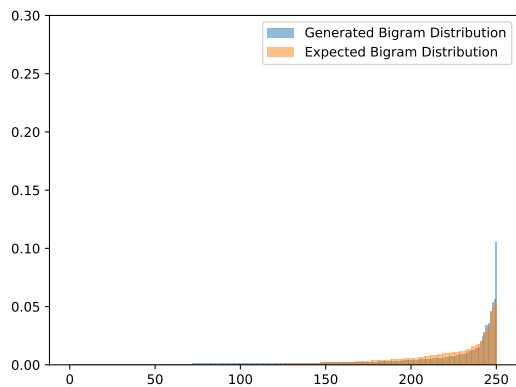
Snapshot 1.0M



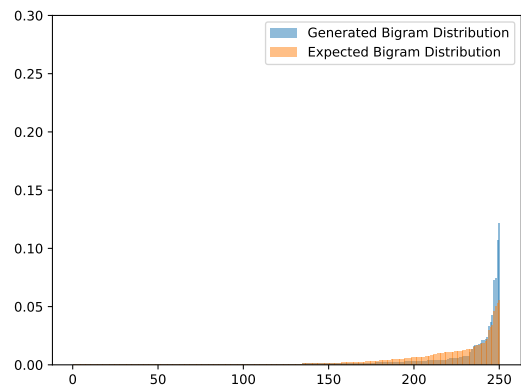
Snapshot 1.5M



Snapshot 2.0M



Snapshot 2.5M

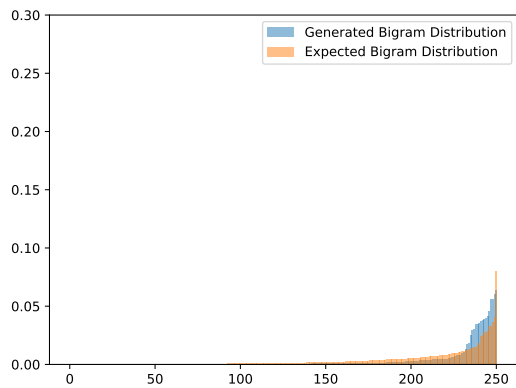


Snapshot 3.0M

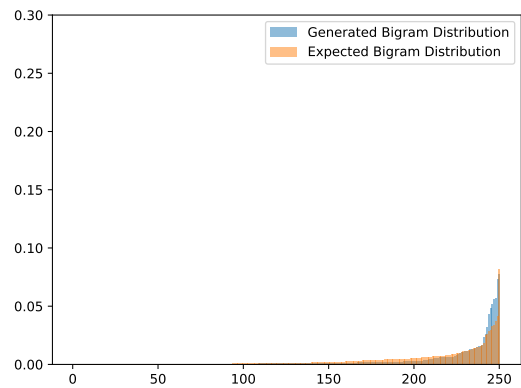
Figure 7.5.: Comparison of the distributions of the top 250 most used bi-grams for the responses of the OpenSubtitles model (orange) when using the test dataset and the distribution within the training dataset (blue). The distributions are compared for each snapshot available.

Reddit The distributions for the Reddit model reveal an interesting development (see Figure 7.6). In the beginning, same as with the OpenSubtitles, the distributions are evenly spread. However, as the training continues, the distributions increasingly better fit the expected distributions, until the fit reaches the highest overlap when using the 1.5M and 2.0M snapshots. From this point onwards, the distributions start to drift apart again. This coincides in our opinion, that the Reddit model is at its performance peak when using the snapshot 2.0M. This is probably associated with the fact that we are doing more than two epochs over the full dataset. From the snapshot 2.0M on, the model itself starts to become worse, as the diversity and quality of the responses starts to decline. We are going to analyze this assumption below.

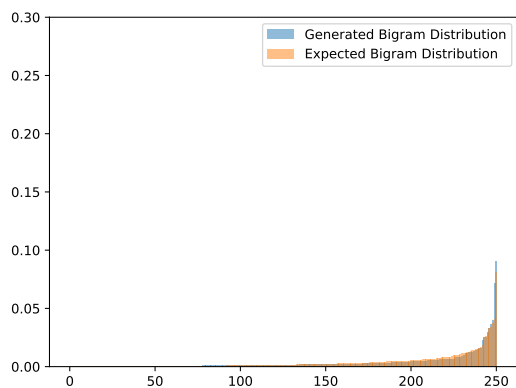
The behavior of the uni-gram distributions develops in a similar way, therefore we documented these visualizations in the Appendix C.



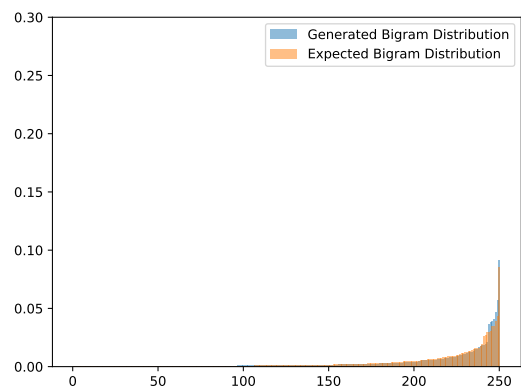
Snapshot 0.5M



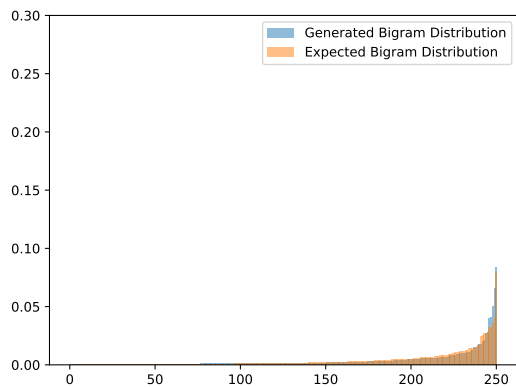
Snapshot 1.0M



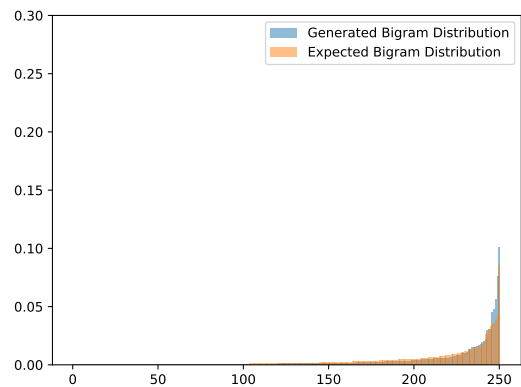
Snapshot 1.5M



Snapshot 2.0M



Snapshot 2.5M



Snapshot 3.0M

Figure 7.6.: Comparison of the distributions of the top 250 most used bi-grams for the responses of the Reddit model (orange) when using the test dataset and the distribution within the training dataset (blue). The distributions are compared for each snapshot available.

7.3.2. Language Diversity

We now analyze the language diversity of the outputs produced by the two models over the different snapshots. For this purpose, we are analyzing how large the share of the top 10 uni-, bi-gram and sentences within the entirety of the generated responses is when using the respective test datasets. The results can be seen in the Table 7.12 for Reddit and 7.11 for OpenSubtitles.

In the Tables, three different categories are shown: Uni-grams, bi-grams and sentences. For each of these categories, we have summed up the occurrence frequencies of the top 10 most occurring instances for the respective category. These sums can be seen in the *Top Count* columns. The columns named *All Count* contain the total amount of all instances for each category per snapshot. The ratio of the *Top Count* value compared to the *All Count* value can be seen in the *Top 10 Share* column for each category and snapshot.

The first thing which is noticeable when comparing the values from the two Tables is that the Reddit outputs seem to be composed of much more uni- and bi-grams. This can be seen when comparing the *Top Count* values from both Tables. This signifies that the outputs of the Reddit model are of greater length on average than the outputs of the OpenSubtitles model because the number of sentences stays the same for both. This might be one of the reasons why the results from the Sent2Vec analysis (see Chapter 7.2) are much better for the Reddit than the OpenSubtitles model.

At first, we tried to see a tendency on how language diversity develops when using the absolute values from the *Top Count* columns, but we could not. This is the reason why we added the relative values from the *Top 10 Share* columns. The problem with using the absolute values is that the number of used uni- and bi-grams changes between the snapshots. However, when we start using the relative values we can compare the different snapshots to each other by means of diversity of the outputs.

Snapshot	Words			Bi-Gramm			Sentences		
	Top 10 Count	All Count	Top 10 Share	Top 10 Count	All Count	Top 10 Share	Top 10 Count	All Count	Top 10 Share
0.5M	516,756	1,035,956	49.88%	306,757	1,033,319	29.69%	102,770	249,984	41.11%
1.0M	639,494	860,951	74.28%	504,946	858,145	58.84%	188,748	249,984	75.50%
1.5M	624,243	1,249,230	49.97%	378,566	1,233,467	30.70%	108,312	249,984	43.33%
2.0M	589,721	814,101	72.44%	475,120	807,087	58.87%	181,585	249,984	72.64%
2.5M	651,779	1,131,640	57.60%	395,060	1,117,436	35.35%	92,273	249,984	36.91%
3.0M	991,879	1,470,695	40.10%	718,034	1,459,490	49.20%	105,245	249,984	42.10%

Table 7.11.: Top 10 uni-grams, bi-grams and sentences with summed up frequencies and computed share of the entirety of all instances for each category per Open-Subtitles snapshot.

Snapshot	Words			Bi-Gramm			Sentences		
	Top 10 Count	All Count	Top 10 Share	Top 10 Count	All Count	Top 10 Share	Top 10 Count	All Count	Top 10 Share
0.5M	2,712,157	4,484,679	60.48%	2,096,681	4,483,620	46.76%	86,408	249,984	34.57%
1.0M	2,576,291	4,294,209	60.00%	1,941,620	4,290,818	45.25%	71,050	249,984	28.42%
1.5M	1,918,226	3,668,416	52.29%	1,256,740	3,663,402	34.31%	46,590	249,984	18.64%
2.0M	2,507,930	4,577,892	54.78%	1,665,799	4,567,229	36.47%	29,544	249,984	11.82%
2.5M	1,623,645	3,148,834	51.56%	1,045,565	3,134,900	33.35%	73,475	249,984	29.39%
3.0M	2,108,447	3,614,679	69.38%	1,449,193	3,599,584	40.26%	82,797	249,984	33.12%

Table 7.12.: Top 10 uni-grams, bi-grams and sentences with summed up frequencies and computed share of the entirety of all instances for each category per Reddit snapshot.

We plot the *Top 10 Share* values for all the categories per model and snapshot to visualize the development of the values for the different categories and possible correlations between them. The resulting graphics can be seen in Figures 7.7 and 7.8. The plots can be interpreted that decreasing values imply that the share of the top 10 most occurring instances per category reduce compared to the whole number of instances of this category. We interpret this as an increase in language diversity, which is positive.

In the mentioned plots, we observe several peculiarities, which we are going to discuss per model below.

OpenSubtitles The plot for the OpenSubtitles (see Figure 7.7) shows the same patterns as with the n-gram distributions seen before. Over the course of the training, it appears that the diversity of the language model wobbles up and down. Our assumption for this flattering development is similar as with the n-gram distributions: Because this occurs only with the OpenSubtitles model and not with the Reddit, it is most probably related to the dataset. The causes could be the missing turn-taking information as mentioned in the in 5.3, or the spoken language. But, if we take a close look at the numbers, we see that the peaks are getting smaller over time. For example, the share of the top 10 sentences and words for the 2.0M snapshot is about 2% to 3% lower than with the 1.0M model. The difference is, with 0.02%, much smaller for the bi-grams. Nevertheless, overall we can see a tendency that the model starts to use a more diverse language model the longer the training advances.

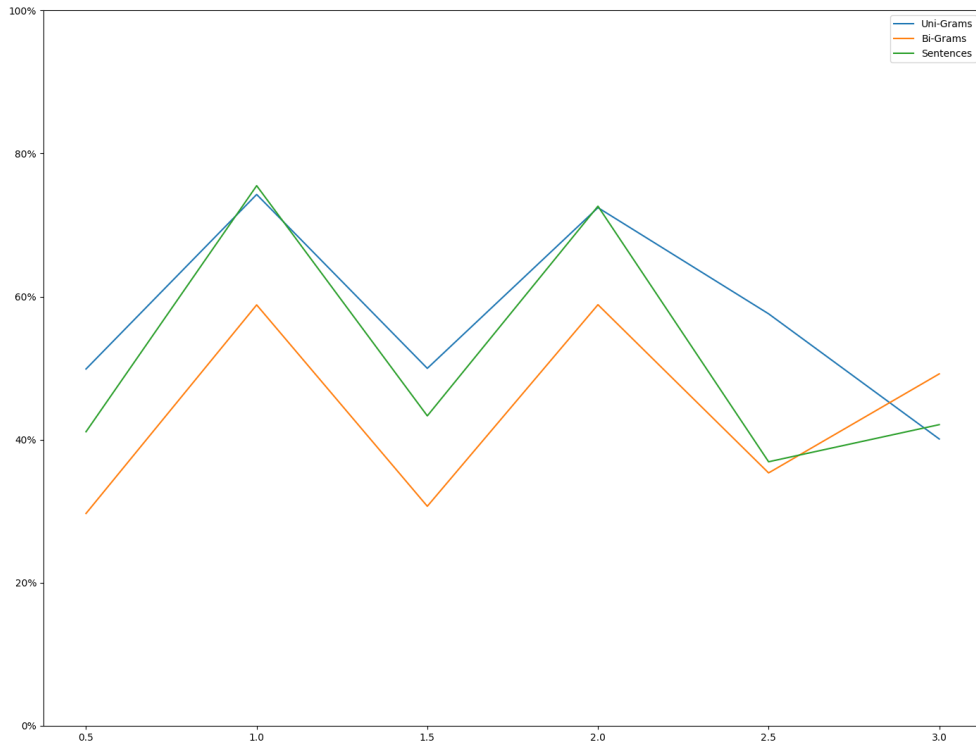


Figure 7.7.: Development of uni-, bi-grams and sentences, in percent, that are covered by the top 10 most used instances of each category for the OpenSubtitles model.

Reddit The plot for the Reddit model (see Figure 7.8) reveals that the language model becomes more diverse over the course of the training until the maximum diversity (minimum in the plot) is achieved when using the 2.0M snapshot. From this point onwards, diversity continuously decreases for the rest of the training. This coincides with our opinion, that the model is most optimal when using the 2.0M snapshot. We assume this problematic behavior is caused by the fact, that between the snapshots 1.5M and 2.0M we start iterating the training dataset the second time. This could lead to an overfitting effect, which worsens the results and diversity of the language used by the model.

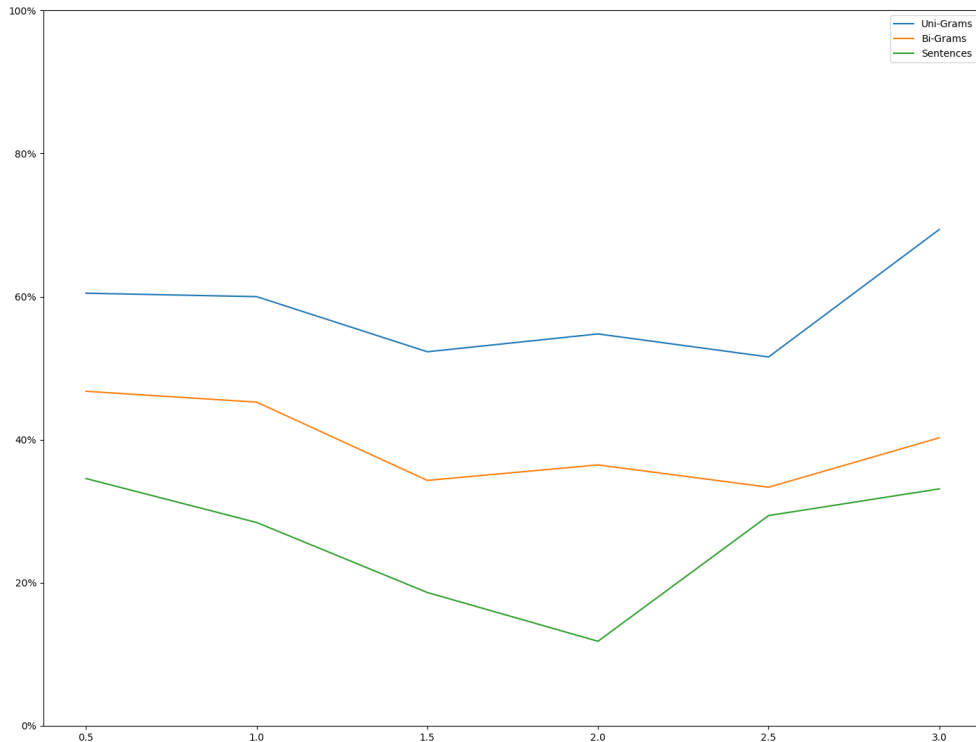


Figure 7.8.: Development of uni-, bi-grams and sentences, in percent, that are covered by the top 10 most used instances of each category for the Reddit model.

7.3.3. Results

Our analysis has shown that the distributions of uni- and bi-grams tend to fit the expected distributions better the longer the training advances. However, we found some interesting peculiarities we wanted to discuss with respect to language diversity.

This analysis has shown that the development of the language diversity is in direct relationship to the development of the mentioned n-gram distributions. We were also able to substantiate our subjective feeling, that the OpenSubtitles model improves over time, as the language diversity increase slowly. Additionally, we could also provide evidence that our opinion is correct that the Reddit model has the best performance when using the 1.5M or 2.0M snapshot as these two snapshots have indeed the greatest variety in the used language measured by our statistics. It does not allow us to completely explain the problem with generic responses, but we could show that the share of these generic responses tends to decrease with more training time.

Finally, the analysis showed that iterating over the same dataset multiple times is not

recommended, as the Reddit model becomes worse after the snapshot 2.0M. This is particularly apparent when looking at the results of the 3.0M snapshot. Using a regularization technique, such as Dropout [47] would probably help when using a small dataset which is iterated multiple times.

With this, we conclude the quantitative analysis of the language model and start the human evaluation. This evaluation is conducted by comparing our model's results against results from others.

7.4. Comparison with Other Models

We now compare the results of our model with two others. As references we use: The CleverBot² chatbot and the results from the paper “Neural Conversational Model” [1], whose model architecture serves as the base of our model (see Chapter 6.1). We test the quality of the responses from our models if confronted with utterances of different types and difficulties. Our expectation is to reveal the strengths and weaknesses of our models.

7.4.1. Cleverbot

We start with the comparison with the CleverBot. CleverBot is a non-deterministic chatbot, meaning similar inputs can create different outputs if fed at different points in time. The responses listed below were generated at the June 1st between 12:00 am and 03:00 pm.

Simple Utterance We start by using the two simple utterances “How are you?” and “Hi there, i m john!”. As one can see in the Tables 7.13 and 7.14, our OpenSubtitles model as well as the CleverBot answered this question with satisfying response. The responses of the Reddit model to the utterance “Hi there, i m john!” are rather poor as none of them is really meaningful.

²<http://www.cleverbot.com/>

Input:	Hi, how are you?
0.5M:	how are you ?
1.0M:	i m fine
1.5M:	i m good
2.0M:	good
2.5M:	good
3.0M:	good
Cleverbot:	I'm good. How are you?

Table 7.13.: Responses from the OpenSubtitles model per snapshot compared to the CleverBot response. The input utterance is “How are you?”.

Input:	Hi there, i m john!
0.5M:	<unknown>
1.0M:	i m not a fan of the <unknown> , but i m not sure if you re a <unknown> .
1.5M:	i m not the only one who thought this was a good idea .
2.0M:	i m a <unknown> !
2.5M:	i m not your buddy , guy .
3.0M:	i m not .
Cleverbot:	So what are you doing?

Table 7.14.: Responses from the Reddit model per snapshot compared to the CleverBot response. The input utterance is “Hi there, i m john!”.

Closed Question In the next step we are comparing responses produced when using a closed question as the utterance. The results are documented in the Tables 7.15 and 7.16. With this type of utterance, we intend to evaluate if the models understand a closed question and can respond appropriately.

Our models can respond to closed questions appropriately, as both of them provide useful answers. The CleverBot seems to have problems with this question, probably because it does not understand how the concepts of flying and birds are related.

Input:	Can birds fly?
0.5M:	no
1.0M:	i can t
1.5M:	yeah , but in the meantime , i can t swim
2.0M:	no
2.5M:	i can t
3.0M:	sure
Cleverbot:	Some say they can.

Table 7.15.: Responses to a contextual question from the OpenSubtitles model per snapshot and the CleverBot response.

Input:	Can birds fly?
0.5M:	i m not sure if this is a movie , but i m not sure if it s a movie .
1.0M:	i can not imagine how much i can get this .
1.5M:	i m sure he s a <unknown> .
2.0M:	yes
2.5M:	yes
3.0M:	yes
Cleverbot:	Some say they can.

Table 7.16.: Responses to a closed question from the Reddit model per snapshot and the CleverBot response.

Knowledge Question The next kind of utterance are knowledge questions. With this kind of questions, we would like to evaluate if the models have general knowledge. The results can be seen in the Tables 7.17 and 7.18.

The responses to the question about the color of the sky show an interesting development. Until the snapshot 2.0M, the responses were not appropriate and did not have any meaningful content³. However, from the snapshot 2.0M on the model answers in a correct way, even though it replies once with red as the answer, which can only be a correct answer in the morning and evening. If we expect blue as the answer, it is also correct twice, whereas the last response from the 3.0M seems to be the best. The answer of the CleverBot to this question is quite meaningless and is not related at all to the provided question.

The response to the question, where the Reddit model lives, was answered in a satisfying way, whereby the response of the 2.0M snapshot is worse than the previous models. It clearly understands the context of the question, as it always answers with a location-related answer. The CleverBot answers this question completely correct.

³The response “the jews” comes from the fact, that in the beginning of the OpenSubtitles dataset, there are a lot of conversations related to movies about Nazism and world war II.

Input:	What color is the sky?
0.5M:	the jews
1.0M:	it s a
1.5M:	oh , it s not
2.0M:	it s the blue
2.5M:	the red
3.0M:	blue
Cleverbot:	*Kisses playfully*.

Table 7.17.: Responses to a knowledge-related question with the OpenSubtitles model per snapshot and the CleverBot response.

Input:	In which country do you live?
0.5M:	i m not sure if you re a <unknown> .
1.0M:	i live in canada .
1.5M:	i live in the uk .
2.0M:	the country
2.5M:	canada
3.0M:	sweden
Cleverbot:	The United States.

Table 7.18.: Responses to a knowledge-related question from the Reddit model per snapshot and the CleverBot response.

Contextual Question With the next question, we want to evaluate if the models are capable of using contextual information to build an appropriate response. The results can be seen in the Tables 7.19 and 7.20.

As visible below, the OpenSubtitles model has major issues with this kind of utterances, as all the responses are not fitting. The only response with certain usability is the one of the 2.5M model, where at least the correct color is included in the response. The response of the CleverBot is in this case also not of relevance.

The responses of the Reddit model are better fitting to the question, but they are also reasonably poor compared to our expectation. At least the model tries to formulate responses where the two colors of the question are included. The answer of the CleverBot is again useless. This time even less related to the question, compared to the response given by the OpenSubtitles model when using the question.

Input:	I wear a blue jeans. What color do my jeans have?
0.5M:	<unknown> <unknown>
1.0M:	i don t know
1.5M:	oh , yeah
2.0M:	i m not a prostitute
2.5M:	i got a blue blue
3.0M:	i m not a fan
Cleverbot:	You don't have any.

Table 7.19.: Responses to a contextual question from the OpenSubtitles model per snapshot and the CleverBot response.

Input:	I wear a red pullover and blue jeans. What color do my jeans have?
0.5M:	i m a big fan of the <unknown> .
1.0M:	i m not a fan of the <unknown> , but i do nt know what you re talking about .
1.5M:	i m a white male and i wear pants .
2.0M:	i wear a red shirt and i wear glasses .
2.5M:	i m a red blue and blue blue .
3.0M:	i m a white male and i have a blue blue shirt .
Cleverbot:	What is your opinion on President Obama. And don't ask me mine without saying yours first!.

Table 7.20.: Responses to a contextual question from the Reddit model per snapshot and the CleverBot response.

Specific Question The next questions are more specific. We are going to ask the OpenSubtitles model a simple mathematical question and the Reddit models a question about the TV series “Game of Thrones”. The results can be seen in the Tables 7.21 and 7.22.

The responses of the OpenSubtitles model are quite clear in the beginning: it does not know the answer or does not understand the question at all. However, the later snapshots (i.e. from 2.0M onwards), it seems to understand that it has to answer with a number. Sadly, the correct result is not included in the generated responses. The CleverBot also does not understand how to answer such a question.

The Reddit model has an interesting twist in its responses. In the beginning, it starts by responding with generic responses which do not seem to have any relation to the TV series. However, the funny response⁴ at the end (i.e. “hodor, hodor, ...”) indicates that it

⁴“Hodor” is a character in the “Game of Thrones” series who is only capable of saying its own name. In case of danger he starts yelling its name over and over again, which occurs several times throughout the series.

understands the question at least to some degree, even though the response does not contain any actual information about the series. The CleverBot response is a mystery to us, as we could not find any character or other scene in the series where a “Dani” is mentioned.

Input:	How much is 1 added to 2?
0.5M:	i don t know
1.0M:	i don t know
1.5M:	i don t know
2.0M:	ten dollars
2.5M:	50
3.0M:	2
Cleverbot:	I don't know.

Table 7.21.: Responses to a mathematical question from the OpenSubtitles model per snapshot and the CleverBot response.

Input:	Can you tell me something about game of thrones?
0.5M:	i m not sure if you re a fan of the movie , but i m not sure if you re going to be a fan of the
1.0M:	i m not sure what you re saying .
1.5M:	i m not sure if i m going to be watching this .
2.0M:	i m not sure if you re joking , but i m pretty sure that s the case .
2.5M:	i m not sure if you re talking about the game or the game , but i think the game is a little different .
3.0M:	hodor hodor
Cleverbot:	Dani is life.

Table 7.22.: Responses to a question about a TV series from the Reddit model per snapshot and the CleverBot response.

Answers to Statements Until now, we have only asked questions to our models. We would now like to see how the responses to statements instead of questions are. The results can be seen in the Tables 7.23 and 7.24.

The responses of the OpenSubtitles model are useful from the snapshot 1.5M onward. Especially the answer for the snapshot 2.0M is interesting, as this seems to indicate that the model has some kind of empathy for the other person in the dialog, expressed in the

response “i m sorry”. The response of the CleverBot signifies that it understands the statement to some degree, but does not know what to answer.

The responses of the Reddit model to the statement that it has a bad opinion are rather poor across all snapshots, with the response from the 1.5M model being the only one which is remotely useful. The response of the CleverBot is a pretty speedy answer in this case.

Input:	i invited her to a public reading of my work , but she never came .
0.5M:	she was a
1.0M:	i don t know
1.5M:	really ?
2.0M:	i m sorry
2.5M:	she s been in a lot of trouble
3.0M:	she s not a fan
Cleverbot:	Who?

Table 7.23.: Responses to a statement from the OpenSubtitles model per snapshot and the CleverBot response.

Input:	Your film taste is bad and your opinion does not interest me!
0.5M:	i m not sure what you re saying .
1.0M:	i m not your friend , but i m not a fan of the <unknown> .
1.5M:	i m sorry , i did nt know that .
2.0M:	i m not a fan of the movie , but i m not sure if you re going to be a fan of the film .
2.5M:	i m not a fan of the film , but i think it s a good movie .
3.0M:	i m not sure if you re being sarcastic or not .
Cleverbot:	And for me your opinion does not matter.

Table 7.24.: Responses to a statement from the Reddit model per snapshot and the CleverBot response.

Self-Concept Finally, we have an interesting answer from the Reddit model to the question “What are you?”. The responses can be seen in Table 7.25. All snapshots produce bad responses, apart from the last, where it answered that it is a bot. This is an interesting answer because it shows that the model seems to potentially have some kind of self-concept. We cannot explain that, as the model was trained on comments from

Reddit discussion about movies and series. Nevertheless, it is an interesting answer to see.

Input:	What are you?
0.5M:	i m not a fan of the movie , but i m a huge fan of the movie .
1.0M:	i m not a fan of the <unknown> , but i m not a fan of the <unknown> .
1.5M:	i m not .
2.0M:	i m not .
2.5M:	i m not .
3.0M:	i m a bot .
Cleverbot:	A human, what are you??

Table 7.25.: Responses to a self-concept question from the Reddit models and the Clever-Bot response.

Sadly, we were not able to find a similar response when using the OpenSubtitles model.

7.4.2. Neural Conversational Model

In this Section, we are going to compare the results from the paper “Neural Conversation Model” (NCM) with the resulting responses from our models. To compare the responses, we are going to use utterances from the paper and show all the responses from all OpenSubtitles snapshots, but only the (subjectively) best response from the Reddit model, as the NCM model was trained on the OpenSubtitles dataset. The responses can be seen in the Tables 7.26 to 7.33. The categories where the input utterances came from in the paper are noted in the Table descriptions. A discussion of the results can be found at the end of this Chapter.

We do not provide individual evaluations to the model responses, as the objective was to gain an overall impression of the quality of responses to individual utterances.

We are only showing a part of the dialogs which were evaluated. All other can be found in Appendix D.

Input:	what year were you born ?
<i>OpenSubtitles</i>	
0.5M:	a lot of people
1.0M:	i was a
1.5M:	a little
2.0M:	i was born
2.5M:	i don t know
3.0M:	i don t know
<i>Reddit</i>	
2.0M:	i was born in the 90s .
<i>NCM:</i>	1977 .

Table 7.26.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “what year were you born ?” from the “Basic”.

Input:	why are we here ?
<i>OpenSubtitles</i>	
0.5M:	because we were here
1.0M:	i don t know
1.5M:	i don t know
2.0M:	i m sorry
2.5M:	we re here
3.0M:	we re here
<i>Reddit</i>	
1.5M:	because we are nt .
<i>NCM:</i>	i 'm not sure .

Table 7.27.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “why are we here ?” from the “Basic” category.

Input:	are you a leader or a follower ?
<i>OpenSubtitles</i>	
0.5M:	no
1.0M:	i m not a
1.5M:	i m sorry
2.0M:	i m not a prostitute
2.5M:	i m a
3.0M:	i m a
<i>Reddit</i>	
2.0M:	i m a cyborg .
<i>NCM:</i>	i'm a leader .

Table 7.28.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “are you a leader or a follower ?” from the “General Knowledge Q&A” category.

Input:	how much is ten minus two ?
<i>OpenSubtitles</i>	
0.5M:	four
1.0M:	ten
1.5M:	two , two , three
2.0M:	ten dollars
2.5M:	ten
3.0M:	ten
<i>Reddit</i>	
2.0M:	i m not sure how much i m going to get downvoted to hell .
<i>NCM:</i>	seventy - two .

Table 7.29.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “how much is ten minus two ?” from the “General Knowledge Q&A” category.

Input:	what is the color of a yellow car ?
<i>OpenSubtitles</i>	
0.5M:	a <unknown> ?
1.0M:	it s a
1.5M:	what ?
2.0M:	it s a blue blue
2.5M:	it s a blue
3.0M:	it s a yellow
<i>Reddit</i>	
2.5M:	the yellow
<i>NCM:</i>	yellow .

Table 7.30.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the color of a yellow car ?” from the “General Knowledge Q&A” category.

Input:	what is the purpose of existence ?
<i>OpenSubtitles</i>	
0.5M:	the jews
1.0M:	i don t know
1.5M:	you re just a fucking boy , man
2.0M:	i don t know
2.5M:	i m sorry
3.0M:	i m not sure
<i>Reddit</i>	
1.5M:	i m not sure if you re joking or not , but i m sure you re not familiar with the word .
<i>NCM:</i>	to find out what happens when we get to the planet earth .

Table 7.31.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the purpose of existence ?” from the “Philosophical Q&A” category.

Input:	what is the purpose of being intelligent?
<i>OpenSubtitles</i>	
0.5M:	the jews
1.0M:	i don t know
1.5M:	i don t know
2.0M:	i don t know
2.5M:	i don t know
3.0M:	i don t know
<i>NCM:</i>	to find out what it is .

Table 7.32.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the purpose of being intelligent?” from the “Philosophical Q&A” category.

Input:	what are the things that i do to be immoral?
<i>OpenSubtitles</i>	
0.5M:	i don t know
1.0M:	i don t know
1.5M:	what s the big idea ?
2.0M:	i don t know
2.5M:	i m a
3.0M:	i m not a criminal
<i>Reddit</i>	
2.5M:	i think you re missing the point . i don 't know .
<i>NCM:</i>	to find out what happens when we get to the planet earth .

Table 7.33.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “what are the things that i do to be immoral?” from the “Morality” category.

7.4.3. Results

In conclusion, our model is not as poor as expected by looking at the problems with the metrics and the generic responses caused by an undiversified language model.

In comparison to the CleverBot, our model provided several responses which were clearly superior in quality. But there were also cases where the CleverBot outperformed our model. This is probably related to the fact that the CleverBot works by fuzzily searching already witnessed conversations. This means, that if a conversation about a certain topic does not exist, the CleverBot has no facilities to create a meaningful answer. In contrast,

our model instead has the ability to “understand” what is said and, hence can at least try to answer. Such an example would be the mathematical question “How much is 1 added to 2?”, where the CleverBot simply answered “I don’t know” and our model started to respond with numbers, even though the result was incorrect.

When comparing our models to the NCM model, one main issue of the model quickly becomes visible: The size of the model. For easily understandable utterances, the responses of our models are almost or as good as the responses from the NCM model. But if the utterances become more complicated, especially when it comes to philosophical or morality questions, our model starts to respond worse than the NCM model. This is, as said, not a real surprise, because our model is only half the size of the NCM model. Nevertheless, we conclude that our model is not superior, but in some regards of comparable quality as the NCM model.

7.5. Beam-Search

There are always some poor responses shown in the previous Chapter. We would like to use our beam-search implementation as described in Chapter 3.3 to evaluate if this can help to increase response quality. We will evaluate this for three different examples, and evaluate if the responses generated by the beam-search decoder are superior to the responses of the greedy decoder. The evaluation was performed subjectively as usually the best answer does not necessarily correlate with the resulting log-probability score. This analysis is exemplarily done with the OpenSubtitles 3.0M model and a beam-width of 200.

As already mentioned, we only list the subjectively best responses here. All other responses can be found in Appendix E. This has to do with the fact, that the subjectively best answer is usually not the best answer found when looking at the log-probability scores.

“what year were you born?” The responses of the greedy decoder to this question can be found in the Table 7.26. The original answers of the OpenSubtitles model were all reasonably poor, and it seemed that the model did not fully understand the question.

However, when using the beam-search decoder, there are several useful responses, such as “1991”, “last year” or “five years ago”. These responses are much better than the results initially generated by the greedy decoder. However, none of them is the highest ranked response and, hence is not returned by the model as the answer of choice. We provide a possible explanation for this anomaly later.

“i wear a blue jeans. what color do my jeans have?” The second question was also answered pretty poorly initially. The results of the greedy decoder can be found in Table 7.19. The best response was delivered by the 2.5M model and was “i got a blue blue”, which, at least, contains the searched color.

When using beam-search, better responses can be found. Such responses include “blue” and “you know that”.

“what is the purpose of being intelligent?” Finally, we wanted to explore a more complicated question from the moral Section of the NCM paper with beam-search. The responses of the greedy decoder can be found in Table 7.32. All responses are of inferior quality and only consist of generic responses, such as “i don t know”.

When using beam-search, several interesting responses were produced, fitting for such a philosophical question, such as “well, it s complicated”, “you can t know” and “well , it s nothing.”

Beam-Search Helps The responses become much more diverse when using the beam-search decoder. The key problem is to choose the best response from the 200 generated responses. Per algorithm, the sum of the logarithmic probabilities at each step in the beam is used. However, this mechanism does not provide the answers we subjectively selected as “best” from the list. We are not quite sure if this is a problem related to the fact that the decoder was trained in a greedy fashion, but response generation was augmented with beam search, which might not be fully compatible. The implementation of beam-search was validated and is correct, as when beam-size is set to 1, answers are the same as when using the greedy decoder. Nevertheless, the implementation provided us insight into the inner mechanics of the model and we were able to find better responses than with the greedy decoder, even though such had to be selected from the list of responses manually.

7.6. Thought Vectors for Input Sequences

After the encoder has processed the entire input sequences, it forwards the thought vector to the decoder to construct the output sequence (see Chapter 3.3). As this is the only direct connection of the encoder to the decoder, the encoder must “encode” all information into this thought vector. Hence, the thought vector hence represents an embedding of an input sequence in an n dimensional vector space, where n stands for the size of the thought vector. If we project the vectors into two dimensions via PCA, sentences with similar meanings should be clustered. That would substantiate that the models have a semantic understanding of the contents of these sentences.

To analyze the embeddings, we collect them for 15 different sample sentences and create

respective thought vectors. We do this with both models and project the vectors via PCA into two-dimensional space. The results of this projection can be seen in Figure 7.9 and 7.10 below.

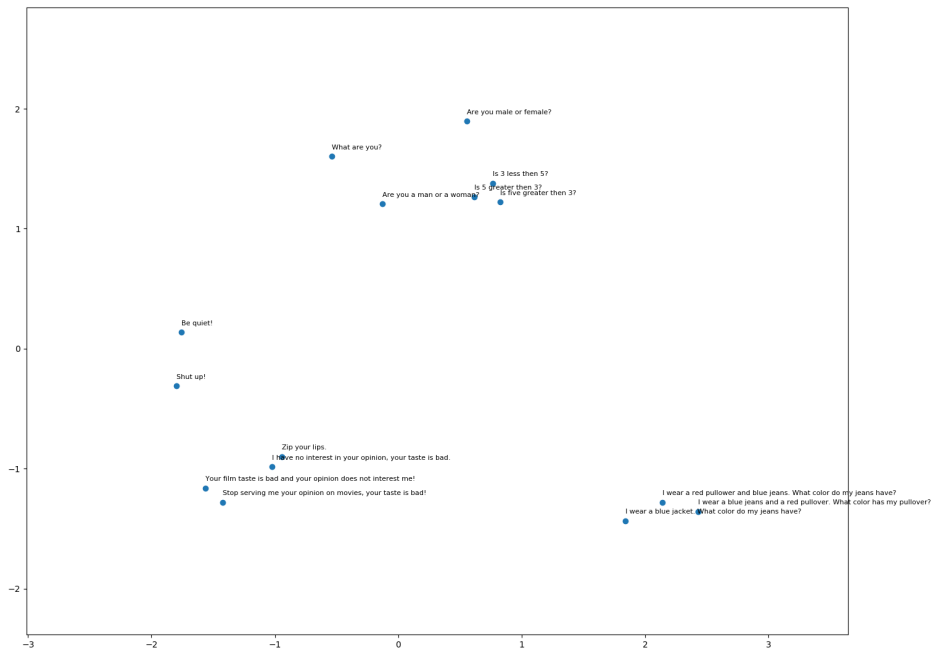


Figure 7.9.: The projected thought vectors for 15 different sentences when using the OpenSubtitles 3.0M model. PCA was used for the projection.

Both of the models seem to have no problems understanding direct sentences where the content is unambiguous (e.g. “I have no interest in your opinion on movies, your taste is bad!”). This can be seen because similar sentences are clustered closely in the projected space. However, when it comes to curses and questions regarding the gender, the OpenSubtitles model starts to struggle, which can be seen by taking a look at the respective points in the projected space. The questions regarding the gender or the curses are scattered throughout the space, even though they should have been embedded closely to each other. The Reddit model seems to have less problems with this, as the embeddings for these sentences are clustered tighter. Interestingly, the Reddit model embeds the sentences with curses close to the sentences regarding the gender.

In general, it appears as the models have an understanding of these different sentences as most of the points are clustered, if they contain similar content. From this, we conclude that our models have, at least to a certain degree, an understanding of the meaning of utterances.

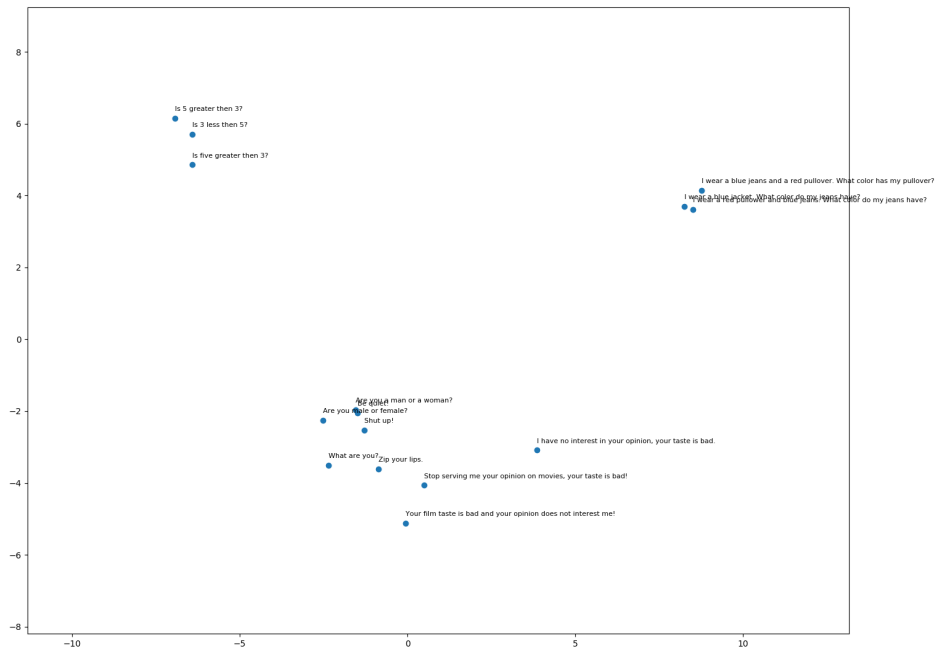


Figure 7.10.: The projected thought vectors for 15 different sentences when using the Reddit 2.0M model. PCA was used for the projection.

7.7. Soft-Attention

In the last part, we take a look at the soft-attention mechanism (see Chapter 3.3) and analyze if the model benefits from its use by looking at the resulting attention weights. We do not expect significant advantages, as Vinyals and Le already notice in their paper [1]. The visualizations are generated by feeding an utterance to our models and then visualizing the resulting attention weights (see Chapter 3.3) in a heat-map over the different time steps of the decoding process. As utterances, we use three different examples, one general question, one mathematical question and an example which consists of two sentences with a relative pronoun in the second sentence, which refers to the subject of the first sentence.

We perform this analysis on the OpenSubtitles 3.0M and the Reddit 2.0M snapshots.

Attention Visualizations The visualizations of the attention weights can be seen in Figures 7.12 and 7.11. Additional visualizations of the same kind are located in Appendix F.

Clearly visible, the generated attention weights do not show a significant alignment with important words from the input utterance. For example, when looking at the input

utterance “anna is 18 years old. how old is she?” (Figure 7.11), we would expect that the decoder would place a large attention weight on the thought vector when the actual age of the person is processed. However, this is not the case for any of the output words.

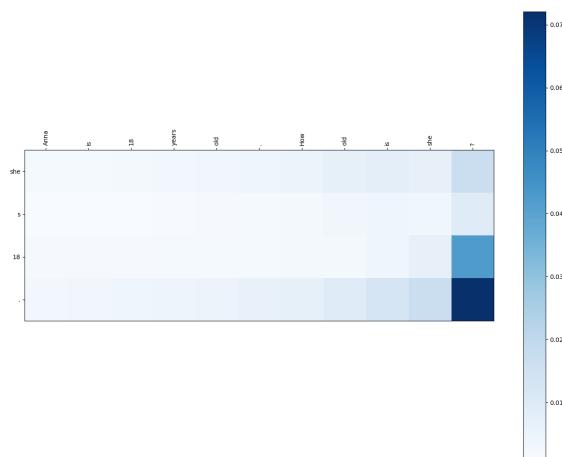


Figure 7.11.: Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.

A second example, where attention does not compute meaningful alignments, is shown in Figure 7.12. Here it is the same problem as before, the attention weights do not align with the important thought vectors from the input utterance.

From this quick analysis, we can reaffirm that the soft-attention mechanism does not seem to help to improve the decoders comprehension in conversational models. Our assumption is that it does not help because there are no clear alignments between the input and output as in other use-cases, such as machine translation or summary generation. This would explain why Vinyals and Le did not see an improvement with regard to loss and perplexity when using the soft-attention mechanism.

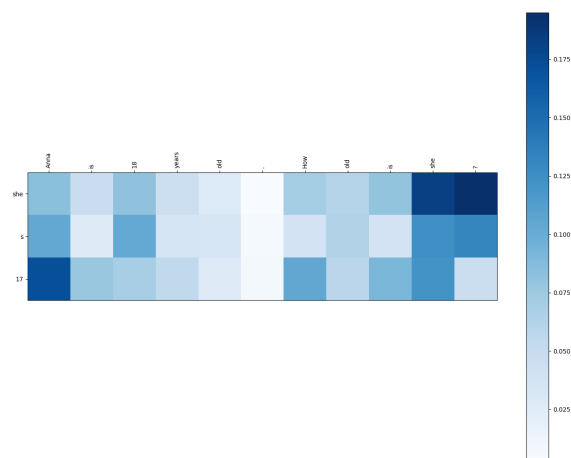


Figure 7.12.: Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.

8. Conclusion

We started this thesis with the goal to build an end-to-end dialog system using the architecture and technologies from the “Neural Conversational Model” paper of Vinyals and Le [1]. Our principle idea was to build such a system, but on a much smaller scale. The goal of this reduction was to fit the resulting system onto a single GPU, instead of using the resources of an entire server as they did in the mentioned paper.

The implementation of our system was challenging, but it allowed us to gain insight on how such systems are implemented on a much lower level and taught us many different things, including how a computational graph works, the inner mechanisms of sequence-to-sequence models. We even had the chance to implement an in-graph beam-search decoder.

Due to difficulties with several different APIs of the `TensorFlow` framework, it took much longer than expected until we could complete our first, running version of the system. But, after all the initial trouble, more than half of the semester has already passed by then, we could finally start the training of our models on the two datasets. We have decided to use this two datasets, namely the Reddit comment and OpenSubtitles dataset, for two reasons: First, our initial goal was to build a system which could talk about movie related topics. The second reason was, that these two datasets represent two different types of language, spoken and written. Handling and preprocessing both datasets was rather challenging, because both are huge datasets; in fact, bigger than anything we have handled before. The Reddit dataset also imposed another difficulty because we had to reconstruct the original discussions by using a tree structure.

After we have obtained both datasets and a functional version of the system, we started to determine the maximum possible size of the model still fitting on a single GPU. It took some time to define all required hyperparameters as they determine the final memory consumption of the model.

When we found the maximum achievable size for the models, we proceeded with the training. To track the development of the models over the course of the training period, we created snapshots of the models at six distinct points in time. After more than three weeks of training time, we had gathered our six snapshots and could start the analysis phase.

At first glance, everything looked fine and the loss and perplexity values on the training datasets indicated was overall successful. There were differences in speed and the way how the models learned over the course of the training. Obvious was the high variance

in the metrics for OpenSubtitles which we had not observed with the Reddit model. We presume that this was related to the differently structured datasets, as the Reddit dataset has a clear structure implied by the nature of the raw data. This was in clear contrast to the OpenSubtitles datasets, where no information about turn-taking or structure of the conversations is available.

We then started the evaluation of our models with the respective test datasets, based on the mentioned cross-entropy loss and perplexity metrics. Additionally, we also explored the usability of a new metric based on the recently published Sent2Vec library. At first, we were negatively surprised by the results and considered the entire experiment has failed. However, we subsequently found several reasons for why the models did not show the expected performance. Our first finding was, that the cross-entropy loss function and the related perplexity metric might not be the most suitable metrics for measuring the performance of a conversational dialog system, as the range of possible appropriate responses is huge. We were aware of this problem from the beginning. Because of this, we also considered a new, alternative metric based on the Sent2Vec library. However, the results of the performance assessment of the models with this metric did also not provide results as expected. We assume, as with the other metrics, the problem lies in the large proportion of generic responses. One interesting fact we derived from the results was, that the Reddit model performed twice as good as the OpenSubtitles model in this evaluation. We concluded this to be related to the differences between spoken and written language, which are both present in our datasets. Overall, the results of the metrics did, in our opinion, not reflect the effective performance of the models because we had already talked to them and considered the model’s responses to be better than the metrics implied. Therefore, we started to investigate into the language models to find the reason for the poor results on the metrics.

Our first attempt to find a reason for the metric’s poor results was to look at the generated responses. We quickly noticed the lack variety in the responses. Instead, our models often generated generic responses, such as “i don t know” and “i m not sure”. We started to investigate into how our models adopted the language models from the training datasets. We did this by looking at the development of uni- and bi-gram distributions over the course of the training. This revealed, that the learning process of the OpenSubtitles model differed from the learning process of the Reddit model. The OpenSubtitles model exhibited much more variance, which we attributed to the much noisier dataset, compared to the Reddit dataset. This analysis also revealed, that it is not recommended to use smaller dataset and iterate over it multiple times, as we did with the Reddit model. The result was, that the later snapshots of the Reddit model showed a much poorer overall performance on metrics. Similar effects were observed in the subjective evaluation, where we talked to the model directly.

We still believed that the generic responses could be explained somehow. Therefore, we conducted another analysis where we analyzed the usage frequencies of the top 10 most used uni-, bi-grams and sentences in the outputs when using the test datasets. Our intention with this analysis was to see how the language variety evolves over time. The results showed that the amount of generic responses decreases over time. We could also confirm our subjective impression that the models tend to improve with extended training time,

especially the OpenSubtitles model. For the Reddit model, we could validate through this analysis that the language variety and hence, the performance of the model is at its peak when using the 1.5M or 2.0M snapshots. Finally, we did not find a conclusive answer to the question what the reasons for the generic answers are. However, we could show that this problem reduces as the models have more time to learn.

After all the analysis, we started to compare our models to two others, namely the CleverBot and the results from the paper mentioned at the beginning of this Chapter. Our models delivered comparable performance as the CleverBot and the results from the paper, if the utterances did not involve complicated or indistinct sentences. In general, the results from the comparison of our models to the result from the paper were disillusioning, because most of the utterances we used from the paper yielded poor results on our side. However, this was not a surprise, as we already knew that the samples from the paper included a lot of complicated topics, such as morality and philosophy. We concluded that the driving factor that the responses by our models were not as expressive and thoughtful as the ones in the paper are the shorter training time and the limited model size.

As the last analysis, we took a quick look into the beam-search implementation, the soft-attention mechanism and the clustering of thought vectors. This revealed, that the results from the beam-search decoder are often better than when using the greedy decoder. The analysis of the soft-attention mechanism and its impact on the generation of responses by our models showed, that it limited usability of the approach in the context of conversational models; as already noticed by others. The clustering of generated thought vectors however showed, that our models indeed had an understanding of language, as most of the sentences with comparable meaning were clustered tightly.

In summary, we are satisfied with the outcome of this thesis, even though the results were not as good as initially expected. It provided us the opportunity to explore into a complete new and previously unknown topic in the machine learning domain. We gained knowledge and insight into many new theories, research areas and architectures, while simultaneously allowing us to implement a system to solve a really interesting challenge. We learned much about the most recent innovations in the field of dialog systems. We would have preferred more time for the elaboration of this thesis, as this would have allowed us to extend the time for the training of the model and even deeper investigation of certain aspects.

9. Future Work

To finish our theses, we would like to give several leads how the developed system and models could be extended in a future work:

- The first, obvious change we would have liked to make is to extend the time used for training. Such models need a lot of time to be trained and as our analysis and our impression has shown, it would be advisable to train the mentioned models for a longer period than only three weeks. We cannot say anything about what the most appropriate training time would be. This is especially true for the OpenSubtitles model as we were only able to process a little more than half of the training dataset. For the Reddit model, the size of the dataset would obviously have to be extended by using a larger portion of the corpus in order that such a long training makes sense.
- The second, maybe obvious point, is to grow the size of the model itself. This is not easily possible, as we already used the largest size possible so that the model still fit on a single GPU. However, the implementation of the models could be extended to support multi-GPU use-cases. This would require a major redesign of the implementation of the models, as the computational graph would have to be split across the multiple GPUs. This is possible in TensorFlow, via the `tf.device` API¹, which allows for running the computations across multiple different devices (i.e. GPUs, CPUs). Such a split was already done by other people, for example in one of the original sequence-to-sequence learning papers by Sutskever et al [9].
- We had to fix the size of our computational graph to use 30 time steps for the encoder and decoder each. This is due to the fact, that we had to use an old TensorFlow API (see Chapter 4.2) because we were running into huge problems when first developing our system. Another, also obvious change, would be to use the dynamic RNN API² from the most recent TensorFlow version to implement the models.
- Another point is to try different RNN cells instead of LSTM. For example, Gate Recurrent Units [24] or even Convolutional LSTM [25] cells could be used. Another idea is to try other extensions to LSTM cells, for example by using peephole connections.
- We located the problem, that our model often generates generic responses. This

¹https://www.tensorflow.org/api_docs/python/tf/device

²https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn

could be tackled by using dedicated loss function which trains the models with an incentive on language variety, such as the loss function found in the paper by Li et al. [48].

- One of the main issues we had were the metrics to assess the performance of such conversational models. We are currently not aware of any other popular metrics used in the field, besides the cross-entropy loss and the perplexity. However, it would be certainly possible and an interesting research topic to develop new metrics, such as we tried with our Sent2Vec metric.
- Another interesting point would be the possibility to formulate the sequence-to-sequence learning as a beam-search optimization problem as Wiseman et al. [49] did.
- A point which we have not covered at all in this thesis is context between the utterances within a conversation. This is a hard problem, as the models would have to store the context somehow between the utterances. Two ways on how to head into that direction are shown in the papers by Yao et al. [50] and Serban et al. [51].

Appendix

A. Neural Networks

The following Chapter contains a quick introduction into neural networks (NN) and the methods for training them, namely backpropagation and gradient-descent.

Neural networks are a model used in the area of machine learning, which is biologically motivated and loosely mimics the function of the human brain. In the following paragraphs, we are going to explain the functions of all the components which are part of a NN.

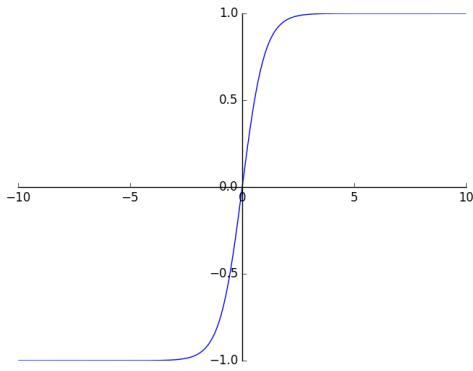
Neuron A NN, at the lowest level, is composed of *neurons*, sometimes also called *perceptrons*. These neurons are basically modeling a mathematical functions and are the building blocks of every NN.

These neurons accept n input values $\mathbf{x} = (x_0, x_1, \dots, x_n)$ and use them to compute a single output value o . A unique weight w_n from the set $\mathbf{w} = \{w_0, w_1, \dots, w_n\}$ is assigned to each of the input values. The input value of x_0 is almost always set to 1 and not further changed; this value is called the *bias* value and allows the modeled function to be affine instead of just linear. This increases the modeling power of such neurons, as the space of functions which can possibly be modeled grows. With the aforementioned input values \mathbf{x} , the associated weights \mathbf{w} and the activation function φ , the output o of the neuron can be computed as shown in equation A.1:

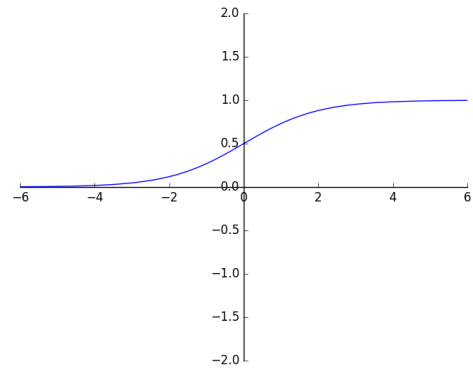
$$o = \varphi(\mathbf{w} \cdot \mathbf{x}) = \varphi\left(\sum_{i=0}^n w_i x_i\right) \quad (\text{A.1})$$

The activation function φ is responsible for squeezing the result of the computation of a neuron into a predefined range of values; for example using *tanh* always results in output values in the range $[-1, +1]$, no matter which scale the input values originally had. Examples of commonly used activation functions are *tanh*, *relu*, *sigmoid* or *binarystep*. They are visualized as plots in Figure A.1.

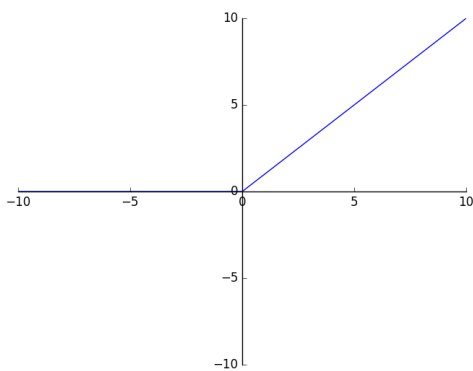
Layer With the neurons introduced one paragraph earlier, we can now start to build the layers of a NN. Each layer consists of multiple neurons stacked on top of each other, as seen in Figure A.2. These layers are then arranged in a sequential manner to form a complete NN. Each NN usually has at least three of these layers: The first one is



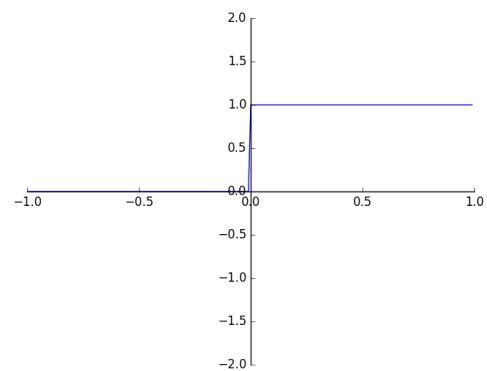
(a) $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



(b) $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$



(c) $\text{relu}(x) = \max\{0, x\}$



(d) $\text{binarystep}(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$

Figure A.1.: Plots of several commonly used activation functions for neurons in NNs.

called the *input layer*, the second the *hidden layer* and the most right one is the *output layer*.

The input data $\mathbf{x} = (x_1, x_2, \dots, x_n)$ enters the NN through the input layer. At this stage, the bias x_0 value is usually set to 1 again. Most of the time, there is only one bias value and weight for all neurons in a layer of an NN. The input values in \mathbf{x} are then forwarded to the neurons of the (first) hidden layer which then compute their activations o_{ln} , where l signifies the layer and n shows the position of the neuron in that layer respectively. The resulting activation values are then passed to the output layer, where the embodied neurons compute their activation values $o_{21}, o_{22}, \dots, o_{2n}$. The input values of a single neuron in the output layer are usually the activation values of all neurons in the preceding layer. Such a layer is also called *fully-connected*, as every neuron uses all values from the preceding layer. The output of the NN is then the set of activation values $o_{31}, o_{32}, \dots, o_{3m}$ in the output layer, where m signifies the number of neurons in the output layer.

This procedure of doing one forward-pass through the NN is called *forward propagation*.

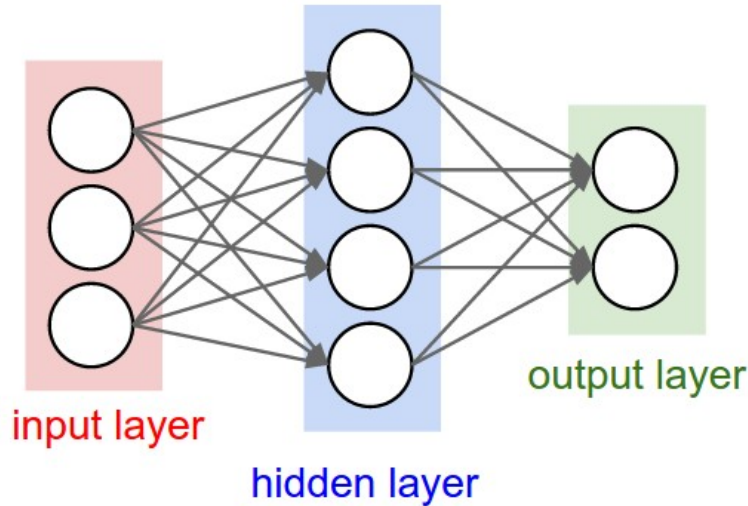


Figure A.2.: Simplified visualization of a NN with one input, hidden and output layer.

Our example only consists of one hidden layer, but one can also imagine NNs with multiple hidden layers; such NNs are then called *deep neural networks*. The number of layers and neurons in each of them is strongly dependent on the nature of the problem it is applied to.

Backpropagation with gradient-descent As in almost all models in the area of machine learning, a NN learns by optimizing a *loss function* L , also sometimes called error function. Every function which can be used to quantify the predictive error of a NN can be used as a loss function. As examples, one could mention the *mean squared deviation* $\text{msd}(y_{\text{true}}, y_{\text{pred}}) = \frac{1}{n} \sum_{i=0}^n (y_{\text{pred}} - y_{\text{true}})^2$ or the one used for the model of this thesis, the *categorical cross-entropy* function $H(p, q) = -\sum_x p(x) \log(q(x))$. The optimization of this loss function is almost always done via a method called *backpropagation* in conjunction with the *gradient-descent* algorithm. The optimization is then done as follows:

1. Do the forward propagation with the given input values $\mathbf{x} = (x_0, x_1, \dots, x_n)$ as described above.
2. Use the predicted and expected values in combination with the defined loss function to quantify the predictive error of the NN.
3. The predictive error is now backpropagated through the NN via the gradient-descent algorithm. The weights of all inputs for each neuron are then adapted with respect to their influence on the exhibited predictive error.

The influence of each weight on the predictive error is determined by computing the partial derivate of the loss function L with respect to the respective weights, as seen in Equation A.2. After computing the partial derivation, the weights are updated accordingly. This is done by multiplying the computed derivation value by the learning rate

η and subtracting the resulting value from the current value of the weight. The learning rate defines, how strong a single run of gradient-descent alters each weight in the network.

The new value for the weight i in layer l , with given learning rate η and loss function L , is then computed as follows:

$$w_{li} := w_{li} - \eta \frac{\delta E(\mathbf{w})}{\delta w_{li}} \quad (\text{A.2})$$

In practice, this computation is done in a vectorized manner to speed up the computation significantly. Here we use the gradient, hence the name, of the loss function with respect to all weights \mathbf{w} :

$$\mathbf{w} := \mathbf{w} - \eta(\nabla_{\mathbf{w}}E(\mathbf{w})) \quad (\text{A.3})$$

One of the big drawbacks of gradient-descent is, that its success is highly dependent on the chosen learning rate η . When the learning rate is chosen too large, the algorithm might miss the optimum or the value of the loss function even diverges; if the learning rate is too small on the other hand, it might take a really long time until the final optimum is found. We've visualized this problem in Figure A.3, where the development of an arbitrary loss function with different learning rates is plotted over time.

To mitigate this issue, there exist several different advancements to gradient-descent, such as *AdaGrad* [45] or *AdaDelta* [52], which adapt the learning rate according to the updates done in the past. An extensive overview of different gradient-descent algorithms and their variations, including the two mentioned before, can be found in [53].

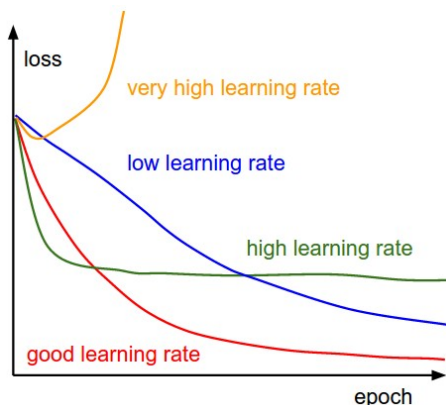


Figure A.3.: Visualization of the development of a loss function when using different learning rates.¹

¹<http://cs231n.github.io/assets/nm3/learningrates.jpeg>

B. Using the Software System

The following chapter gives a tutorial on how to use the software system (see Chapter 4) implemented to conduct the experiments in this thesis.

B.1. Download

The code of the software system can be downloaded by using `git`¹. The repository is located on the publicly accessible GitHub website². However, the visibility of the repository is set to private. To gain access, please send an e-mail with the request and the username of your GitHub account to either `vongrdir@students.zhaw.ch` or `weilemar@students.zhaw.ch`.

B.2. Requirements

To use the software and all the related scripts, the following software packages must be installed:

- `python` in the version 3.5.2³
- If a GPU is used:
 - Nvidia GPU driver⁴ for the respective GPU.
 - Nvidia `cuda` 8 toolkit⁵.

The experiments and script can also be conducted without a GPU and solely on the CPU. However, this leads to higher runtime in several parts of the system, mainly when training models. In addition to the mentioned software packages the `python` libraries listed in Chapter 4.7 must be installed.

¹<https://git-scm.com/>

²<https://github.ch/vongrdir/BA-ML-17>

³<https://www.python.org/>

⁴<http://www.nvidia.de/Download/index.aspx>

⁵<https://developer.nvidia.com/cuda-toolkit>

The libraries can be installed via the `python` package manager `pip`⁶. It is recommended to use the exact versions of the mentioned software packages and `python` libraries to avoid any compatibility issues. However, it might certainly be possible that the software system runs fine with newer version without any problems.

⁶<https://packaging.python.org/installing/>

B.3. Structure of the Repository

In the following table, we explain the structure of the repository and the contents of the main directories.

Name of Directory	Description
<code>configs/</code>	In this directory, all the JSON configurations for all experiments are stored.
<code>report/</code>	All documents related to the thesis are stored in this directory.
<code>misc/</code>	Directory for storing miscellaneous files.
<code>results/</code>	The results of all the run experiments are stored in this directory. The results themselves are stored in a directory named after the WHAT of the experiment. In total, the model, all collected metrics and the configuration of experiments are stored in this directories.
<code>scripts/</code>	The scripts used in this thesis are stored in this directory (see Chapter B.4).
<code>source/</code>	The whole source code of the software system itself is located in this directory including all the components necessary to implement the system described in the Chapter 4.

Table B.1.: Remarks regarding the structure of the repository.

B.4. Using the Scripts

In the following section, we are going to introduce the most important scripts necessary to use the software system for running experiments. The scripts themselves are located in the directory `scripts/`.

Name	Description
<code>analyse_ngram_from_corpus.py</code>	With this script, it is possible to analyse a corpus regarding its bigrams. This script was used to generate the n-grams in Chapter 5.7 and 7.3.
<code>analyze_timestamp_problematic_opensubtitles.py</code>	With this script, the time-lag analysis between utterances in the OpenSubtitles corpus can be done (see Chapter 5.6).
<code>analyze_word_coverage.py</code>	This script allows to analyse the word coverage of a corpus with regard to a given vocabulary (see Chapter 5.4).
<code>split_corpus.py</code>	This script allows to split a given corpus into a training, test and validation set by proportions. The split itself is done randomly (see Chapter 5.5).
<code>evaluate_trained_model.py</code>	This script allows to evaluate a trained model on a certain dataset and stores the resulting metrics of this evaluation.
<code>generate_s2v_sequence_embeddings.py</code>	This script allows for generating Sent2Vec embeddings for the given list of sequence samples. These are then used in Chapter 7.2 for the similarity analysis.
<code>get_internal_embeddings_from_samples.py</code>	This script allows to generate thought vectors for a list of given sample texts. They are used in the analysis in Chapter 7.6.
<code>preprocess_opensubtitles_data.py</code>	This script is responsible for preprocessing the raw OpenSubtitles corpus as described in Chapter 5.3.
<code>preprocess_reddit_corpus.py</code>	This script is responsible for preprocessing the raw Reddit corpus and building the tree which is finally converted into the datasets as described in Chapter 5.3.
<code>talk_to_model.py</code>	This script provides a possibility to talk to trained models through the terminal. The functionality is basically the same as in the web frontend (see Chapter B.6).

Table B.2.: Descriptions of the most important scripts to use the software system.

There are much more scripts than the ones described above (e.g. for plotting, creation of vocabularies). Not all of them are necessary to conduct experiments, which is why only explain the most important.

In the following table there are exemplary calls for all scripts listed above:

Name	Exemplary Call
<code>analyse_ngram.py</code>	<code>python scripts/analyse_ngram.py data/opensubtitles/opensubtitles_raw.txt 2 results/opensubtitles/bigram_analysis.csv results/opensubtitles/bigram_analysis_words.csv</code>
<code>analyse_timestamp_problematic_opensubtitles.py</code>	<code>python scripts/analyse_timestamp_problematic_opensubtitles.py data/OpenSubtitles2016 analysis_timestamps_opensubtitles.json</code>
<code>analyse_word_coverage.py</code>	<code>python scripts/analyse_word_coverage.py data/reddit/reddit_corpus.txt word_coverage_reddit_new.json data/reddit/vocab_100k.pickle data/reddit/vocab_50k.pickle</code>
<code>split_corpus.py</code>	<code>python scripts/split_corpus.py data/reddit/reddit_corpus_preprocessed.txt 80,10,10 data/reddit/reddit_train.txt data/reddit/reddit_valid.txt data/reddit/reddit_test.txt</code>
<code>evaluate_trained_model.py</code>	<code>python scripts/evaluate_trained_model.py results/reddit/model-100000.chkp data/reddit/reddit_train.txt results/reddit/test_metrics.json results/reddit/test_predictions.csv 250000</code>
<code>generate_s2v_sequence_embeddings.py</code>	<code>python scripts/generate_s2v_sequence_embeddings.py misc/fasttext misc/sent2vec_wiki_bigrams 700 2 results/reddit/test_predictions.csv results/reddit/test_s2v_generated_wiki_bigrams.h5</code>
<code>get_internal_embeddings_from_samples.py</code>	<code>python scripts/get_internal_embeddings_from_samples.py samples.txt results/reddit/model-100000.chkp results/reddit/samples_embeddings.h5</code>
<code>preprocess_opensubtitles_data.py</code>	<code>python scripts/preprocess_opensubtitles_data.py data/opensubtitles/raw-xml-files/ data/opensubtitles/opensubtitles_raw.txt</code>
<code>preprocess_reddit_corpus.py</code>	<code>python script/preprocess_reddit_corpus data/reddit/full_corpus/ 2014,2015 movies</code>
<code>talk_to_model.py</code>	<code>python scripts/talk_to_model.py</code>

Table B.3.: Exemplary calls for the most important scripts to use the software system.

B.5. Running Experiments

To run experiments, one has to write a configuration file in the JSON format. All of them reside in the directory `configs/`. In the following table, all important configuration parameters are explained:

Name	Default Value	Description
<code>device</code>	<code>/gpu:0</code>	Defines the device on which the computations are run.
<code>train</code>	<code>true</code>	Defines whether a training should be run or not. Is set to <code>false</code> in case inference is run.
<code>git_rev</code>	<code>null</code>	This parameter stores the SHA1 of the latest <code>git</code> revision when an experiment was started.
<code>model_path</code>	<code>null</code>	This parameter signifies if an already trained model should be loaded before starting the training or inference.
<code>training_data</code>	<code>null</code>	Defines which dataset should be used for training the model while training.
<code>validation_data</code>	<code>null</code>	Defines which dataset should be used for validating the model while training.
<code>reverse_input</code>	<code>false</code>	Defines whether the input sequence should be fed to the model in reverse if set to <code>true</code> .
<code>use_last_output_as_input</code>	<code>false</code>	Defines whether the output of the last sample should be used as the input for the next sample.
<code>start_training_from_beginning</code>	<code>false</code>	Defines whether the training should start from the first sample of the training dataset or not. If it is set to <code>true</code> , the training will start with the first sample the model has not already seen in previous trainings, as indicated by internal variables of the model saved when storing the model.
<code>show_predictions_while_training</code>	<code>false</code>	Defines whether the outputs of the model should be printed to the terminal when training.
<code>show_predictions_while_training_num</code>	<code>5</code>	Defines how much predictions should be printed at the end of each epoch when training.
<code>vocabulary</code>	<code>null</code>	Defines which <code>pickle</code> vocabulary should be used for the current model.
<code>epoch</code>	<code>1</code>	Defines the number of epochs which should be done in case of training a model.
<code>save_model_after_n_epochs</code>	<code>10</code>	Defines the interval in which the model is stored based on the number of epochs finished.
<code>epochs_per_validation</code>	<code>10</code>	Defines the interval in which the model is validated while training based on the number of epochs finished.
<code>batches_per_validation</code>	<code>0</code>	Defines the interval in which the model is validated while training based on the number of epochs finished.
<code>batches_per_epoch</code>	<code>1000</code>	Defines how much batches should be processed in one epoch.
<code>batch_size</code>	<code>1</code>	Defines how much samples should be put in one batch.

Table B.4.: Explanation of the important configuration parameters of the software system (part 1).

Name	Default Value	Description
<code>max_input_length</code>	50	Defines the maximum number of words to consider from the input sequence.
<code>max_output_length</code>	50	Defines the maximum number of words the decoder can generate before the decoding stops.
<code>num_encoder_layers</code>	1	Partially defines the number of layers to use for the encoder and decoder. This number is added to <code>num_decoder_layers</code> to get the full number of layers for the encoder and decoder.
<code>num_decoder_layers</code>	1	Partially defines the number of layers to use for the encoder and decoder. This number is added to <code>num_encoder_layers</code> to get the full number of layers for the encoder and decoder.
<code>num_hidden_units</code>	1024	Defines the size of the hidden state used in the encoder and decoder cell.
<code>cell_type</code>	LSTM	Defines which kind of RNN cell is used for the model. Can be either LSTM, GRU or RNN.
<code>sampled_softmax_number_of_samples</code>	512	Defines the number of words sampled when using the sampled softmax loss function. Is disabled in case the number of words is smaller than the provided number or if the parameter is set to 0.
<code>hidden_state_reduction_size</code>	null	Defines the dimensionality the hidden state of the decoder cell should be projected down to before it is fed to the softmax layer at the end.
<code>max_random_embeddings_size</code>	512	Defines the dimensionality of the word embeddings used within the model.
<code>use_beam_search</code>	false	Defines whether the beam search decoder should be used instead of the greedy decoder.
<code>beam_size</code>	10	The number of beams which should be considered when using the beam search decoder.
<code>beam_search_only_best</code>	true	Defines that the output of the beam search decoder should only be the one resulting from the best beam if set to true, or the results from all beams otherwise.
<code>buckets</code>	[[50, 50]]	Defines the buckets which are used when using the model. We only used one bucket, but it is certainly possible to use multiple of them. The range of available buckets has to cover <code>max_input_length</code> and <code>max_output_length</code> .
<code>dropout_input_keep_prob</code>	1.0	Defines how much percent of the input to the RNN cells should be kept when using dropout.
<code>dropout_output_keep_prob</code>	1.0	Defines how much percent of the output of the RNN cells should be kept when using dropout.

Table B.5.: Explanation of the important configuration parameters of the software system (part 2).

One peculiarity can be seen from the names of the configuration parameters: Some of them contain the word *epoch*, but not necessarily refer to the definition of epoch found in the glossary. This is related to the way training works with TensorFlow, as it does use the term epoch differently. Instead of referring to one full iteration over the training corpus, it refers to a fixed number of batches being processed, because TensorFlow itself has no notion of epoch but rather uses the term *step*. A fixed number of steps is then called an epoch.

The descriptions above are neither complete nor concluding. We recommend consult the source code for the exact behavior of the additional parameters. Also, there are more model related parameters which are only applicable to the models stored in the source code file `other_models.py`, where implemented, but now unused models reside.

An example configuration can look as follows:

```
1  {
2    "epochs": 150000,
3    "batches_per_epoch": 100,
4    "batches_per_validation": 2500,
5    "epochs_per_validation": 50,
6    "save_model_after_n_epochs": 100,
7    "batch_size": 64,
8    "cell_type": "LSTM",
9    "num_hidden_units": 2048,
10   "hidden_state_reduction_size": 1024,
11   "num_encoder_layers": 1,
12   "num_decoder_layers": 1,
13   "training_data": "data/reddit_train.txt",
14   "validation_data": "data/reddit_valid.txt",
15   "vocabulary": "data/reddit/vocab_50k.pickle",
16   "max_random_embeddings_size": 1024,
17   "max_input_length": 30,
18   "max_output_length": 30,
19   "reverse_input": false,
20   "show_predictions_while_training": true,
21   "buckets": [[30, 30]],
22   "sampled_softmax_number_of_samples": 512,
23   "word_tokenizer": "none",
24   "use_last_output_as_input": false,
25   "start_training_from_beginning": false
26 }
```

Figure B.1.: Example JSON configuration for the Reddit experiment.

To start experiments, one has to invoke the `run.sh` script at the root of the repository with the configuration of the experiment desired to run.

```
$ ./run.sh configs/config-1.json
```

The results are then stored in a directory in `results/` within a subdirectory named after the name of the configuration file supplied to `run.sh`.

B.6. Web Frontend

We implemented a simple web frontend for communicating with already trained models. It is implemented by using `flask`, `jQuery`⁷ and the `bootstrap`⁸ frontend framework. To use the frontend, one has to start it using the following command from the root of the project:

```
$ python scripts/web/app.py
```

This will start the web frontend running on `localhost` and using the port `9001`. After it has been started, one has to select the model it would like to load from the models in the dropdown at the top. All models found in the `results/` directory (which can be loaded) are listed there. After the selection, a session has to be started by clicking on the *Start* button. This might take a moment, as the loading of the model is a costly process. After the model has been loaded (as indicated by ...), one can start to communicate with it by sending text. Keep in mind, that the kind of output of the model (e.g. beam-search or greedy) directly depends on the configuration stored in the directory where the loaded model is located. This means, if one wants to see for example the output of all beams (in case of beam-search), one has to change the configuration value of the key `beam_search_only_best` to `false` there. For a comprehensive list of all configuration values and their meaning, please see Chapter B.5 or the source code itself.

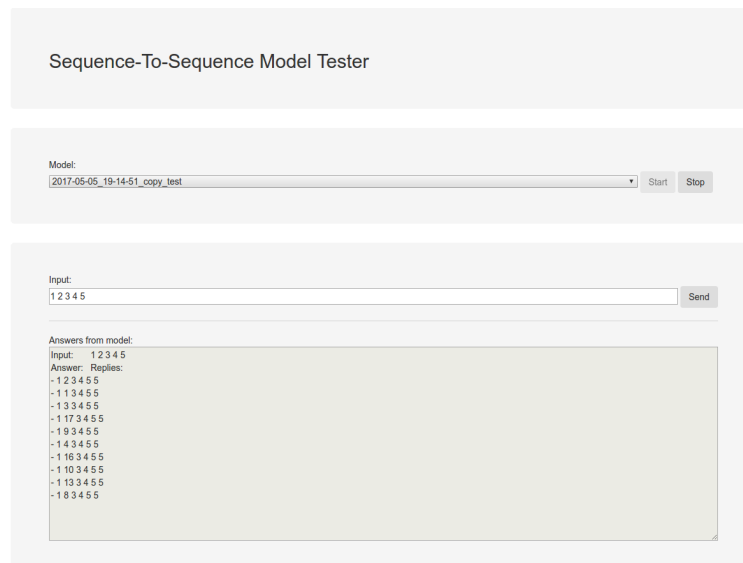


Figure B.2.: Frontend showing the output when sending the sequence “1 2 3 4 5” to a model trained on the copy task (see chapter 4.3).

One last remark: Currently, there is still a bug when the user tries to end a running

⁷<https://jquery.com/>

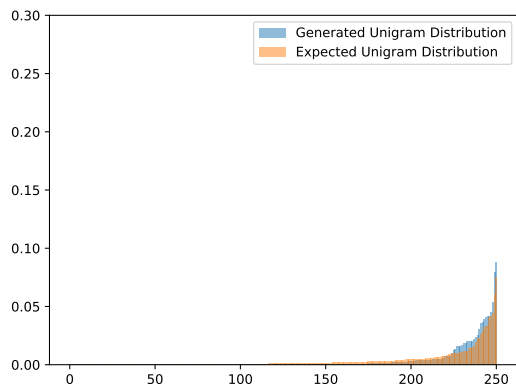
⁸<http://getbootstrap.com/>

session and start a new one. The work around is to simply restart the entire web application.

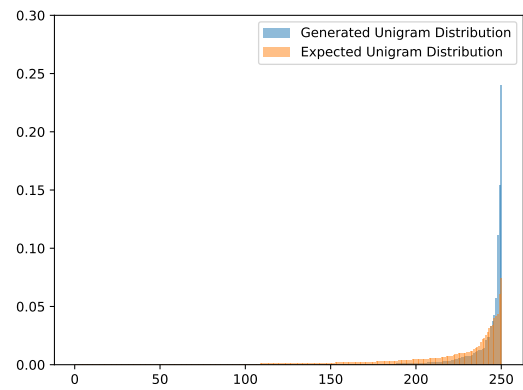
C. Uni-Gram Distributions over Time

Below, you can find the distribution of uni-grams extracted from the output of the model when evaluating it on the test dataset (for each snapshot), compared to the expected distributions of the same uni-grams from the training corpus. They are used in the Chapter 7.3.

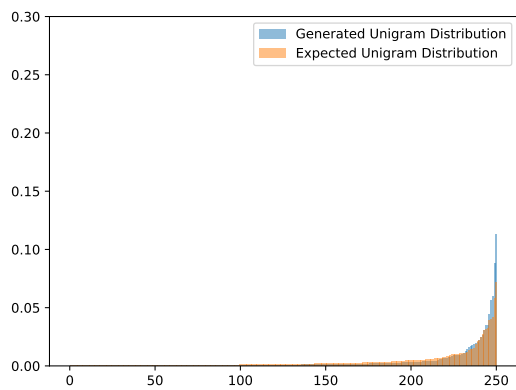
C.1. OpenSubtitles



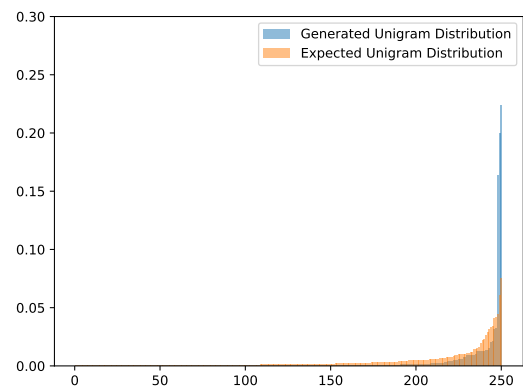
Snapshot 0.5M



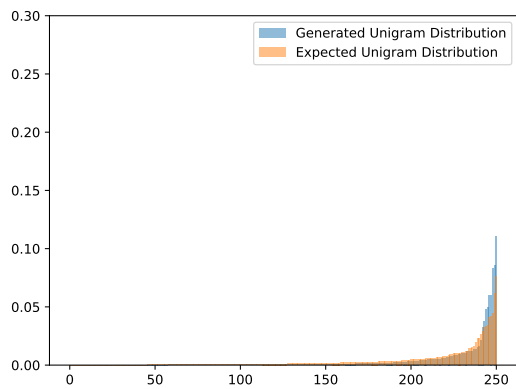
Snapshot 1.0M



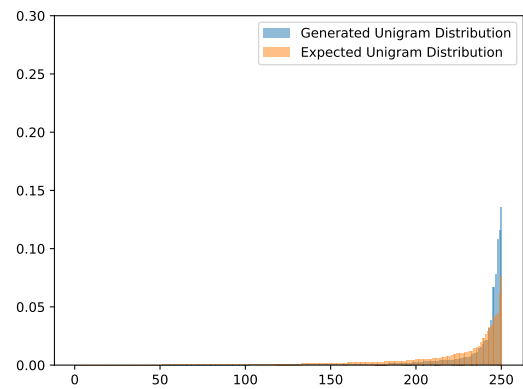
Snapshot 1.5M



Snapshot 2.0M



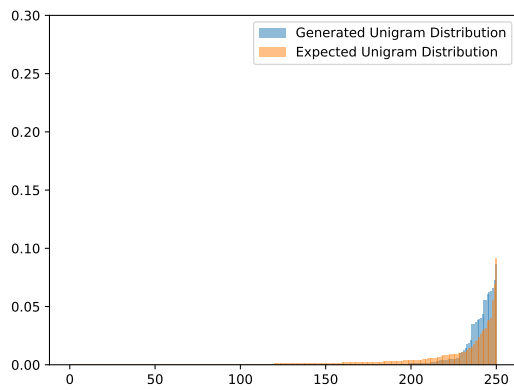
Snapshot 2.5M



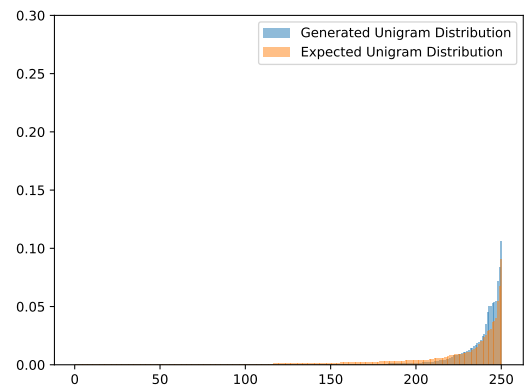
Snapshot 3.0M

Figure C.1.: Comparison of the distributions of the top 100 most used unigrams for the responses of the OpenSubtitles models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.

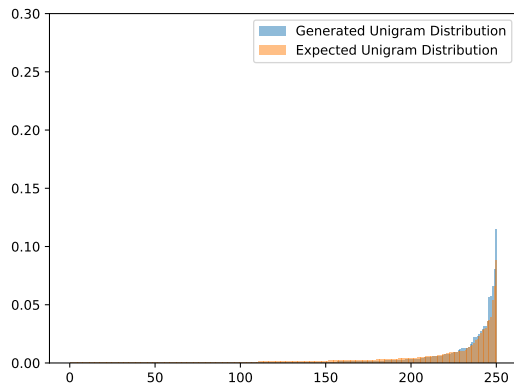
C.2. Reddit



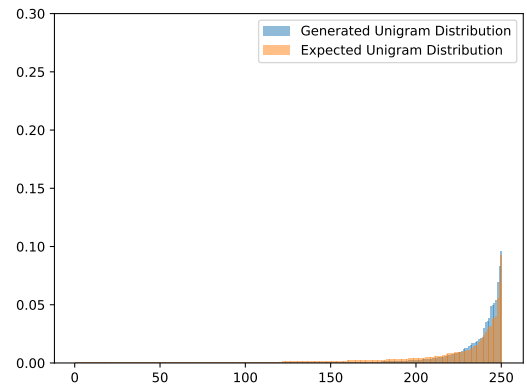
Snapshot 0.5M



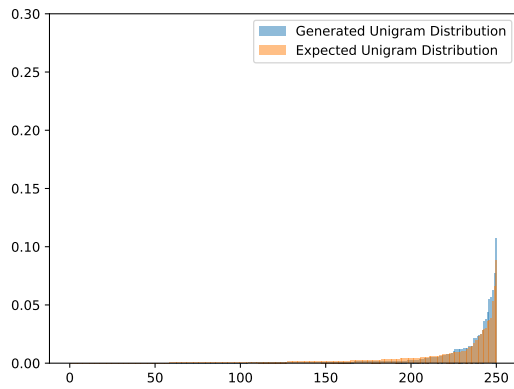
Snapshot 1.0M



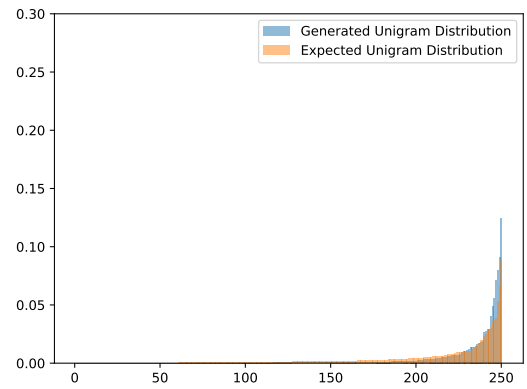
Snapshot 1.5M



Snapshot 2.0M



Snapshot 2.5M



Snapshot 3.0M

Figure C.2.: Comparison of the distributions of the top 100 most used unigrams for the responses of the Reddit models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.

D. Results Compared with Neural Conversational Model

Input:	see you later .
--------	-----------------

<i>OpenSubtitles</i>	
0.5M:	bye
1.0M:	bye
1.5M:	bye
2.0M:	bye
2.5M:	yeah
3.0M:	bye

<i>Reddit</i>	
3.0M:	i m not sure i understand .

<i>NCM:</i>	bye .
-------------	-------

Table D.1.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “see you later.” from the “Basic” category in the paper.

Input:	who is skywalker ?
--------	--------------------

OpenSubtitles

0.5M:	the jews
1.0M:	i don t know
1.5M:	who s the guy who s the boss ?
2.0M:	i m sorry
2.5M:	you re a human
3.0M:	i m not sure

Reddit

2.5M:	the guy who played the game .
-------	-------------------------------

NCM:

	he is a hero
--	--------------

Table D.2.: Comparison between the NCM model and our OpenSubtitles model when using the utterance “who is skywalker?” from the “General Knowledge Q&A” category in the paper.

Input:	does a cat have a wing ?
--------	--------------------------

OpenSubtitles

0.5M:	yes
1.0M:	yeah
1.5M:	yeah
2.0M:	no
2.5M:	a
3.0M:	i m not a fan

Reddit

2.5M:	i think it s a cat .
-------	----------------------

NCM:

	no
--	----

Table D.3.: Comparison of the responses between the NCM model and our models when using the utterance “does a cat have wings?” from the “General Knowledge Q&A” category in the paper.

Input:	what is the purpose of emotions ?
<i>OpenSubtitles</i>	
0.5M:	the jews
1.0M:	i don t know
1.5M:	i m sorry , randal
2.0M:	i don t know
2.5M:	it s a war
3.0M:	i don t know
<i>Reddit</i>	
2.0M:	it s a metaphor for the purpose of the film .
<i>NCM:</i>	i don't know .

Table D.4.: Comparison of the responses between the NCM model and our models when using the utterance “what is the purpose of emotions?” from the “Philosophical Q&A” category in the paper.

Input:	what is moral ?
<i>OpenSubtitles</i>	
0.5M:	it s a
1.0M:	i don t know
1.5M:	what ?
2.0M:	i m sorry
2.5M:	you re a
3.0M:	i don t know
<i>Reddit</i>	
2.5M:	i think it s a good idea to be a doctor .
<i>NCM:</i>	what empowered humanity , what intellectual the essence is .

Table D.5.: Comparison of the responses between the NCM model and our models when using the utterance “what is moral?” from the “Morality” category in the paper.

Input: tell me the definition of morality , i am quite
 upset now !

OpenSubtitles

0.5M: i am

1.0M: i m not gon na tell you

1.5M: i am truly ridiculous , so be my

2.0M: i m sorry

2.5M: i m not going to tell you anything

3.0M: i m not afraid of you

Reddit

2.0M: i m not sure if you re trolling or not , but i m
 not sure if you re trolling or not .

NCM: i'm not ashamed of being a philosopher!

Table D.6.: Comparison of the responses between the NCM model and our models when using the utterance “tell me the definition of morality, i am quite upset now!” from the “Morality” category in the paper.

E. Beam-Search Results with OpenSubtitle

Resulting responses from all 200 beams when using the beam-search decoder on the Open-Subtitles model. See Chapter 7.5 for more information.

E.1. Results for "what year were you born ?"

- 12
- 18
- five
- seven
- birth
- five years
- september 14th
- huh ?
- no
- well
- <unknown >
- life
- the first
- retired
- 42
- 16
- war
- i don t
- fourteen years
- 17 years
- 20
- 11
- four
- 35
- 17
- missouri
- um
- what year
- 25
- nebraska
- 15
- oh
- 19
- mm
- 15 years
- <unknown ><unknown >?
- 5
- what ?
- sixteen years
- twentyeight
- 18
- seventeen years ago
- cambridge
- 27
- 18 years ago
- years ago
- 14 years ago
- history
- 24
- eighteen years ago
- death
- 21
- 17 years ago
- 40
- seven years
- england
- 2009
- yes
- twelve
- zero
- 28
- <unknown ><unknown >
- 30
- uh
- 17
- ten years
- an angel
- family
- 1991
- 16
- 22 years
- 46
- 21
- six
- i
- nine years
- sighs
- my name
- eighteen
- 20 years
- eighteen years
- a couple
- twenty years
- a week
- seventeen years ?
- ten
- hmm ?
- eleven years
- 27
- <unknown ><unknown ><unknown >
- eight years
- eight
- last year
- no
- 1987
- year
- nineteen
- 36
- four years
- 33
- sixteen

- a
- 35 years
- years
- fourteen
- you don t
- 14
- fifteen
- the
- 16 years
- nine
- 14 years ago
- 25 years
- seventeen
- thirteen
- 23
- 67
- 13
- 35
- eleven
- an hour
- 18 years
- two
- 92
- zero
- 15
- 32
- 22
- five years ago
- a year
- 32 years
- 35
- 16
- seventeen
- six years
- seven
- 29
- 32
- fifteen years
- 15
- 28 years
- 31
- 40 years ?
- this year
- a month
- born
- new year
- 17 years
- 25
- me ?
- hmm
- 26
- i
- age
- 13 years
- june
- 27 years
- england
- <unknown
- ><unknown
- ><unknown
- >
- 25 years
- fifteen
- seven years
- the
- seven
- eleven
- ten years
- thirteen
- 16
- six years
- an hour
- huh ?
- mm
- 30
- new year
- i don t
- ten
- eighteen years
- six
- 40 years ?
- 27
- 24
- nineteen
- born
- 27
- 13 years
- england
- 18 years ago
- 5
- 13 years
- 22
- <unknown
- ><unknown
- >?
- what year
- sighs
- <unknown >
- 29
- 22
- four
- 21
- 32 years
- 15

E.2. Results for "i wear a blue jeans. what color do my jeans have?"

(Input): I wear a blue jeans. What color do my jeans have?

(Answer): Replies:

- oh
- what ?
- hey
- yeah
- it s good
- blue
- i m
- i don t
- uh , you <unknown >?
- what are you ?
- no
- okay
- oh , too
- mmhmm
- they re no ?
- it s a
- okay
- <unknown>?
- that s wrong
- it s not me ?
- <unknown >
- uh
- i don t ,
- i don t , ?
- here you
- huh ?
- i mean nothing
- do you ?
- i know
- nothing
- i m
- it s a
- i m
- i ve
-
- groans
- wow
- hey
- all right
- yellow
- all right
- nothing
- ah

- scoffs ><unknown
- uh ><unknown
- green >
- blue ● don t
- i give me ● you re
- hey ! ● is it easy
- i need you ? ● nothing
- shit ● hmm
- give me ● hmm ?
- it s just ● sighs
- yeah ● are you
- okay ● ooh
- oh <unknown >? ● i have
- i m ● red
- mmhmm ● is it
- oh , you know ● oh , uh
- there s ● huh ?
- i ll see ● you don t
- i don t not ? ● oh
- i ve ● it s not me
- no ● hmm
- nothing ● i don t , <unknown >?
- uh , you ● green
- hmm ? ● oh !
- sorry ● give me
- is it ● they re no ?
- sighs ● oh
- okay ● no
- <unknown well , uh
- ><unknown really ?
- ><unknown okay
- > ● it s gon na
- mmm ● nothing
- huh ? ● hmm
- really ? ● oh
- hey , uh ● sighs
- hmm ● it s
- i think it ? ● oh , you know
- what is well ?
- hey ! ● um
- what are you ● no !
- <unknown ● hmm ?
- sorry ● i didn t
- that s that ● it s
- hey ! ● i don t
- um ● i got it s
- what are you ? ● red
- blue ● shit
- <unknown >? ● i don t
- mmm ● nothing
- do you ? ● really ?
- i have ● what ?
- hmm ● i m
- nothing ● what ?
- hmm ● nothing
- it s ● ah
- i m ? ● hey , uh
- <unknown ● blue
- ><unknown ● nothing
- ><unknown ● i m
- > ● what ?
- <unknown >? ● what are you
- laughter ● ooh
- laughs ● oh , too
- let me ● oh , you know
- hey ● ooh
- wow ● it s
- you know that ● mmhmm
- ooh ● blue
- shit ● hmm
- well ● nothing
- what ? ● let me
- i m right ● i m
- it s a good ●
- i ve ● nothing
- i need you ? ● what are you ?
- oh , uh ● hmm ?
- what is well ● i got it s
- well ● it s not me
- oh , you know ● <unknown >?
- mmm ● mmm
- huh ? ● give me
- i didn t , ?
- what are you
- <unknown >?

E.3. Results for "what is the purpose of being intelligent?"

(Input): what is the purpose of being intelligent?

(Answer): Replies:

- | | | | |
|--------------------|--------------------|---------------------|---------------------|
| • i don t | • excuse me | • fine | to |
| • i don t | • well , uh | • i don t | • well ? |
| • you know ? | • i don t , dean ? | • i can do you | • i don t |
| • you are you | • no , uh | • of course you | • i don t |
| know | • i haven t | • i don t | • why ? |
| • nothing | • i don t , actu- | • what are you | • you know ? |
| • nothing | ally | • no | • why ? |
| • what ? | • me ? | • you don t | • i don t |
| • the | • this is | be concerned | • well , um |
| • good | • i m this | <unknown >? | • i don t right |
| • of course you | • sighs | • i m | • i don t just |
| • it s not know | • you know ? | • it s | • well |
| • nothing | • me ? | • please | • good |
| • chuckles | • i | • mm | • well <un- |
| • <unknown | • it s fine | • i don t | known >? |
| ><unknown | • please | • uh | • oh |
| > | • i don t | • i don t | • i don t |
| • me ? | • i don t | • chuckles | • it s |
| • man | • i don t | • i don t | • fine |
| • we have you | • i need | • mm | • what ? |
| • what ? | • i m ? | • i don t | • what are you |
| • i don t , uh | • i don t , c | • i don t right | • i m |
| • we re | • no ! | • what ? | • no |
| • i don t , h | • you can t | • well | • meaning ? |
| • well , it s <un- | • huh ? | • i don t | • no ! |
| known >? | • i don t | • i don t | • you know ? |
| • what is | • i don t | • why ? | • i don t have |
| • nothing | • exactly | • exactly | you |
| • i don t need | • not anymore | • i don t | • you know ? |
| • i don t | • what ? | • no | • i am i supposed |
| • you know ? | • we re | • you know ? | to me ? |
| • they re out | • what ? | • no , uh | • hmm ? |
| there | • i don t | • i don t | • i don t |
| • well , it s com- | • i don t | • well , it s noth- | • good |
| licated | • i don t | ing | • i don t |
| • i don t ! | • you know ? | • all right | • gabriel |
| • you know ? | • what is | • you know ? | • exactly |
| • well ? | • it s not trust | • i don t <un- | • it s not yourself |
| • well , i m | me ? | known >? | • i don t |
| • i don t , sam ? | • you can t know | • i am i supposed | • i don t , you ? |

- | | | | | |
|---------------|----------|-------------------|-------------------|--------------------|
| • sam | | known >? | • what is | ><unknown |
| • well | <un- | • well | • you know ? | > |
| | known >? | known >? | • i don t | • it s |
| • i don t | | • you know ? | • i don t | • i don t |
| • i don t | | • i don t | • you know ? | • you know ? |
| • me ? | | • what ? | • i don t | • i don t |
| • no | | • you don t | • i don t | • i m |
| • i don t | | be concerned | • i don t | • good |
| • i don t | | <unknown >? | • what are you | • i don t |
| • i don t | | • you know ? | • what is | • i don t |
| • you know ? | | • i can do you | • please | • scoffs |
| • <unknown >? | | • you know ? | • nothing | • you are you |
| • scoffs | | • i don t | • i don t , sam ? | know |
| • fine | | • i don t | • of course you | • i don t , dean ? |
| • i don t | | • i don t | • the | |
| • i don t | | • i don t , actu- | • i don t | |
| • well | <un- | ally | • <unknown | |

F. Soft-Attention Visualizations

Below you can find additional visualizations of the attention weights as described in Chapter 7.7.

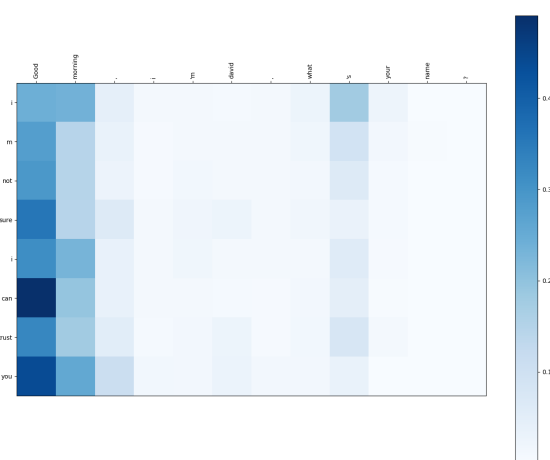


Figure F.1.: Visualization of the attention weights when using the utterance “good morning, i m david, what’s your name?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.

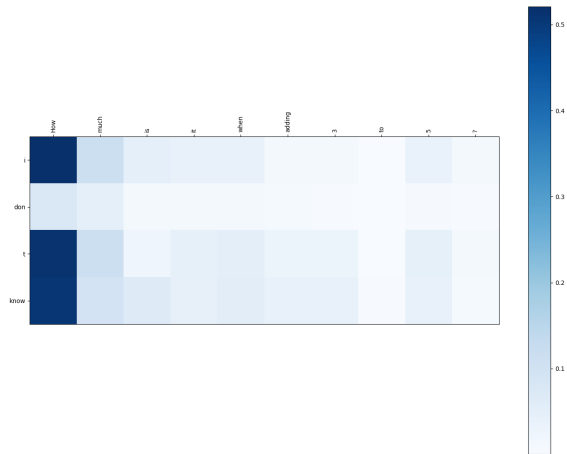


Figure F.2.: Visualization of the attention weights when using the utterance “how much is it when adding 3 to 5?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.

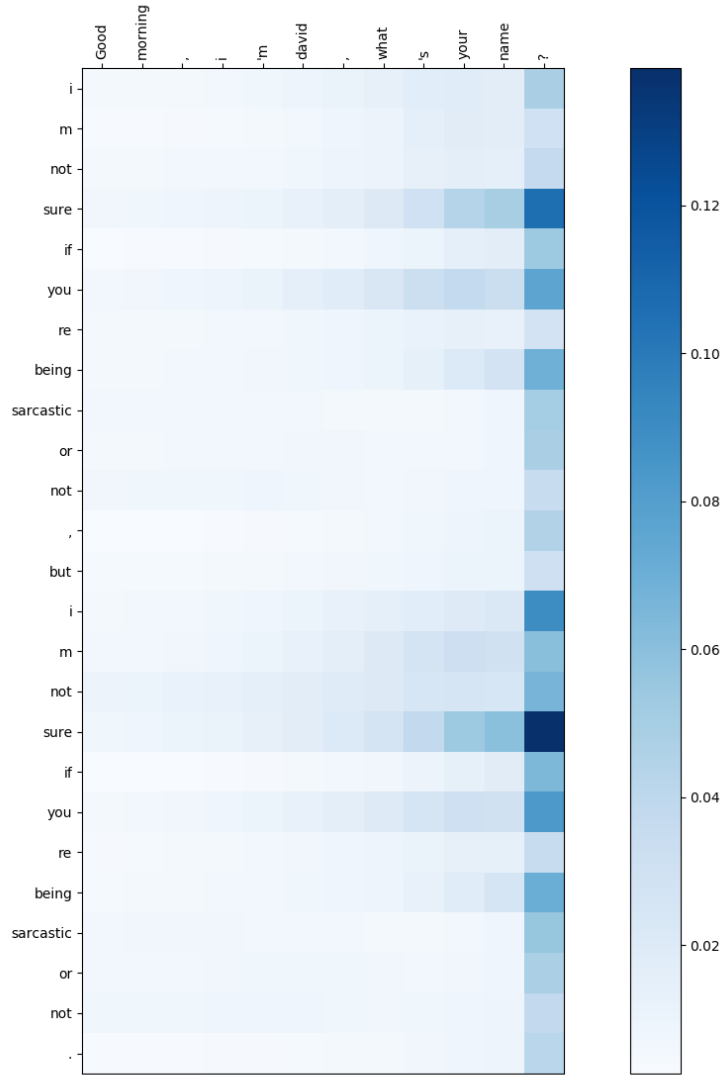


Figure F.3.: Visualization of the attention weights when using the utterance “good morning, i m david, what s your name?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder used on the thought vector of the corresponding word on the x-axis. The Reddit 2.0M was used here.

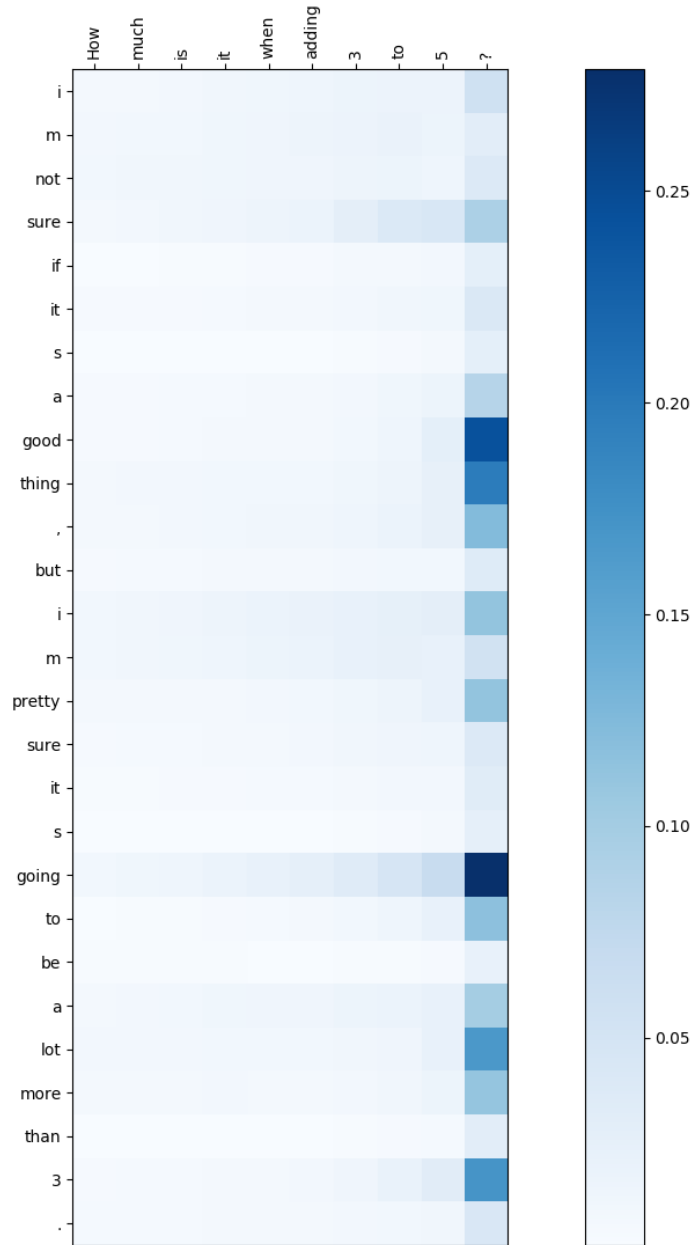


Figure F.4.: Visualization of the attention weights when using the utterance “how much is it when adding 3 to 5?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.

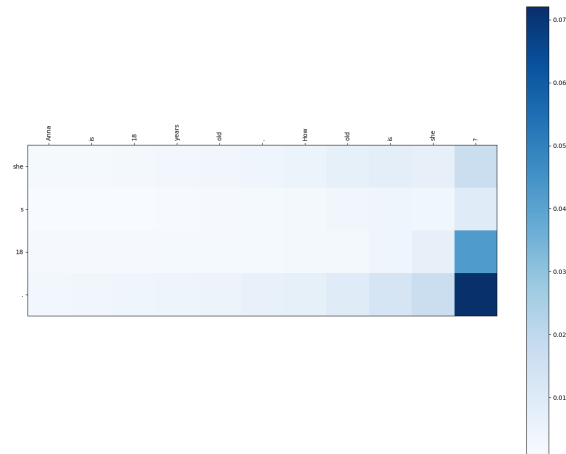


Figure F.5.: Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.

Glossary

Cell	Any instance of any kind of RNN layer is called a cell.
Corpus	Refers to the raw data used to build the datasets.
Dataset	Refers to the dataset resulting from the preprocessing of the raw corpus.
Decoder	Part of seq2seq models responsible for generating the output sequence.
Encoder	Part of seq2seq models responsible for processing the input sequence.
Epoch	One iteration through all of the training data is called an epoch.
Inference	The process of generating predictions with seq2seq models is also called inference.
Layer	Collection of multiple neurons. An NN usually consists of several of these layers.
Neuron	Basic building blocks of NN. Models a mathematical function.
Unit	A single element of the hidden state is called unit.

Acronyms

CPU Central Processing Unit.

GPU Graphical Processing Unit.

LSTM Long Short-Term Memory Network.

NCM Neural Conversational Model (Paper by Vinyals and Le [1]).

NN Neural Network.

RNN Recurrent Neural Network.

Seq2Seq Sequence-To-Sequence.

List of Tables

4.1. Python libraries used in the system.	34
5.1. Origin and some additional information about the raw corpora.	35
5.2. Word coverage of differently sized vocabularies extracted from the generated datasets.	40
5.3. Proportion in which the datasets were split to obtain a train, validation and test dataset.	42
6.1. Hyperparameters used for our seq2seq models.	47
7.1. Loss and perplexity values for each snapshot of the OpenSubtitles model when evaluating it with the test dataset.	53
7.2. Loss and perplexity values for each snapshot of the Reddit model when evaluating it with the test dataset.	53
7.3. Two exemplary dialogs with the Reddit model, one response per snapshot.	54
7.4. Two exemplary dialogs with the OpenSubtitles model, one response per snapshot.	54
7.5. The average similarities when using the Sent2Vec metric on the expected and generated responses from the OpenSubtitles model per snapshot. . . .	56
7.6. The average similarities when using the Sent2Vec metric on the expected and generated responses from the Reddit model per snapshot.	57
7.7. Top 10 most generated responses with respective occurrence frequencies when using the last OpenSubtitles snapshot and the test dataset.	58
7.8. Top 10 most generated sentences with respective occurrence frequencies when using the last Reddit snapshot and the test dataset. The meaning of the <unknown> token is described in 5.4.	58
7.9. The average similarities when applying the Sent2Vec metric on the expected and generated responses on the test dataset when filtering out the top n most generated responses for the OpenSubtitles model.	59
7.10. The average similarities when applying the Sent2Vec metric on the expected and generated responses on the test dataset when filtering out the top n most generated responses for the Reddit model.	59
7.11. Top 10 uni-grams, bi-grams and sentences with summed up frequencies and computed share of the entirety of all instances for each category per OpenSubtitles snapshot.	64
7.12. Top 10 uni-grams, bi-grams and sentences with summed up frequencies and computed share of the entirety of all instances for each category per Reddit snapshot.	65

7.13. Responses from the OpenSubtitles model per snapshot compared to the CleverBot response. The input utterance is “How are you?”	69
7.14. Responses from the Reddit model per snapshot compared to the CleverBot response. The input utterance is “Hi there, i m john!”	69
7.15. Responses to a contextual question from the OpenSubtitles model per snapshot and the CleverBot response.	70
7.16. Responses to a closed question from the Reddit model per snapshot and the CleverBot response.	70
7.17. Responses to a knowledge-related question with the OpenSubtitles model per snapshot and the CleverBot response.	71
7.18. Responses to a knowledge-related question from the Reddit model per snapshot and the CleverBot response.	71
7.19. Responses to a contextual question from the OpenSubtitles model per snapshot and the CleverBot response.	72
7.20. Responses to a contextual question from the Reddit model per snapshot and the CleverBot response.	72
7.21. Responses to a mathematical question from the OpenSubtitles model per snapshot and the CleverBot response.	73
7.22. Responses to a question about a TV series from the Reddit model per snapshot and the CleverBot response.	73
7.23. Responses to a statement from the OpenSubtitles model per snapshot and the CleverBot response.	74
7.24. Responses to a statement from the Reddit model per snapshot and the CleverBot response.	74
7.25. Responses to a self-concept question from the Reddit models and the CleverBot response.	75
7.26. Comparison between the NCM model and our OpenSubtitles model when using the utterance “what year were you born ?” from the “Basic”.	76
7.27. Comparison between the NCM model and our OpenSubtitles model when using the utterance “why are we here ?” from the “Basic” category.	76
7.28. Comparison between the NCM model and our OpenSubtitles model when using the utterance “are you a leader or a follower ?” from the “General Knowledge Q&A” category.	77
7.29. Comparison between the NCM model and our OpenSubtitles model when using the utterance “how much is ten minus two ?” from the “General Knowledge Q&A” category.	77
7.30. Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the color of a yellow car ?” from the “General Knowledge Q&A” category.	78
7.31. Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the purpose of existence ?” from the “Philosophical Q&A” category.	78
7.32. Comparison between the NCM model and our OpenSubtitles model when using the utterance “what is the purpose of being intelligent?” from the “Philosophical Q&A” category.	79

7.33. Comparison between the NCM model and our OpenSubtitles model when using the utterance “what are the things that i do to be immoral?” from the “Morality” category.	79
B.1. Remarks regarding the structure of the repository.	98
B.2. Descriptions of the most important scripts to use the software system.	99
B.3. Exemplary calls for the most important scripts to use the software system.	100
B.4. Explanation of the important configuration parameters of the software system (part 1).	101
B.5. Explanation of the important configuration parameters of the software system (part 2).	102
D.1. Comparison between the NCM model and our OpenSubtitles model when using the utterance “see you later.” from the “Basic” category in the paper.	110
D.2. Comparison between the NCM model and our OpenSubtitles model when using the utterance “who is skywalker?” from the “General Knowledge Q&A” category in the paper.	111
D.3. Comparison of the responses between the NCM model and our models when using the utterance “does a cat have wings?” from the “General Knowledge Q&A” category in the paper.	111
D.4. Comparison of the responses between the NCM model and our models when using the utterance “what is the purpose of emotions?” from the “Philosophical Q&A” category in the paper.	112
D.5. Comparison of the responses between the NCM model and our models when using the utterance “what is moral?” from the “Morality” category in the paper.	112
D.6. Comparison of the responses between the NCM model and our models when using the utterance “tell me the definition of morality, i am quite upset now!” from the “Morality” category in the paper.	113

List of Figures

3.1.	Internal structure of a vanilla RNN cell. ¹	16
3.2.	Internal structure of a LSTM cell [21].	19
3.3.	Internal structure of a Sequence-To-Sequence Model. ²	20
3.4.	Example on how attention can be visualized using a heatmap [26]. The input sequence is on the y-axis and the respective response on the x-axis. Each column in the heat-map can be interpreted as the probability distribution over the thought vectors of the encoder which the decoder accesses at each decoding step.	23
3.5.	Visualization on how beam search works in the context of seq2seq models. ³	25
4.1.	Frontend showing the output when sending the sequence “1 2 3 4 5” to a model trained on the copy task (see Chapter 4.3).	32
5.1.	Example XML entry from the OpenSubtitles corpus.	36
5.2.	Example JSON entry from the Reddit corpus.	38
5.3.	Example of an utterance before and after the preprocessing has been applied.	39
5.4.	Exemplary tree how the comment tree is structured for a single thread. A stands for the threads root node and B to F are comments.	39
5.5.	Semi-logarithmic histogram showing the percentage of words missing per utterance for specific vocabulary sizes.	41
5.6.	Relative distribution (discrete) of the time-lag between two utterances in the raw OpenSubtitles corpus. Most of the utterances lie in the range from 1 to 5 seconds: 13.0% within 1 second, 26.4% within 2 seconds, 22.8% within 3 seconds, 13.4% within 4 seconds and 7.0% within 5 seconds. . . .	43
5.7.	Occurrence frequencies of the 30 most used bi-grams in the OpenSubtitles (left) and Reddit (right) datasets.	44
6.1.	Image for illustrating unrolling an RNN over a fixed size of time steps. ⁴ . .	46
7.1.	Development of the loss and perplexity values on the training and validation datasets throughout the training of the OpenSubtitles model. One tick on the x-axis is equal to 100 batches processed.	51
7.2.	Development of the loss and perplexity on the training and validation datasets throughout the training of the Reddit model. One tick on the x-axis is equal to 100 batches processed.	52
7.3.	Results of the evaluation with Sent2Vec on the outputs of the OpenSubtitles model when using the test dataset. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity for each snapshot.	56

7.4.	Results of the evaluation with Sent2Vec on the outputs of the Reddit model when using the test dataset. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity for each snapshot.	57
7.5.	Comparison of the distributions of the top 250 most used bi-grams for the responses of the OpenSubtitles model (orange) when using the test dataset and the distribution within the training dataset (blue). The distributions are compared for each snapshot available.	61
7.6.	Comparison of the distributions of the top 250 most used bi-grams for the responses of the Reddit model (orange) when using the test dataset and the distribution within the training dataset (blue). The distributions are compared for each snapshot available.	63
7.7.	Development of uni-, bi-grams and sentences, in percent, that are covered by the top 10 most used instances of each category for the OpenSubtitles model.	66
7.8.	Development of uni-, bi-grams and sentences, in percent, that are covered by the top 10 most used instances of each category for the Reddit model.	67
7.9.	The projected thought vectors for 15 different sentences when using the OpenSubtitles 3.0M model. PCA was used for the projection.	82
7.10.	The projected thought vectors for 15 different sentences when using the Reddit 2.0M model. PCA was used for the projection.	83
7.11.	Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.	84
7.12.	Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.	85
A.1.	Plots of several commonly used activation functions for neurons in NNs.	93
A.2.	Simplified visualization of a NN with one input, hidden and output layer.	94
A.3.	Visualization of the development of a loss function when using different learning rates. ⁵	95
B.1.	Example JSON configuration for the Reddit experiment.	104
B.2.	Frontend showing the output when sending the sequence “1 2 3 4 5” to a model trained on the copy task (see chapter 4.3).	105
C.1.	Comparison of the distributions of the top 100 most used unigrams for the responses of the OpenSubtitles models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.	108

C.2.	Comparison of the distributions of the top 100 most used unigrams for the responses of the Reddit models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.	109
F.1.	Visualization of the attention weights when using the utterance “good morning, i m david, what’s your name?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.	119
F.2.	Visualization of the attention weights when using the utterance “how much is it when adding 3 to 5?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The OpenSubtitles 3.0M was used here.	120
F.3.	Visualization of the attention weights when using the utterance “good morning, i m david, what s your name?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder used on the thought vector of the corresponding word on the x-axis. The Reddit 2.0M was used here.	121
F.4.	Visualization of the attention weights when using the utterance “how much is it when adding 3 to 5?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.	122
F.5.	Visualization of the attention weights when using the utterance “anna is 18 years old. how old is she?”. On the x-axis, the input utterance is placed at the top of the chart from left to right. On the y-axis, the response from the model is placed from top to bottom. Each square in the heatmap corresponds to the attention weight the decoder computed for the thought vector of the corresponding word (x-axis) when producing the corresponding response word (y-axis). The Reddit 2.0M was used here.	123

Bibliography

- [1] O. Vinyals and Q. Le, “A neural conversational model,” *ArXiv preprint arXiv:1506.05869*, 2015.
- [2] A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, 1950.
- [3] P. Lison and J. Tiedemann, “Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles,” in *Proceedings of the 10th International Conference on Language Resources and Evaluation*, 2016.
- [4] M. Pagliardini, P. Gupta, and M. Jaggi, “Unsupervised learning of sentence embeddings using compositional n-gram features,” *ArXiv preprint arXiv:1703.02507*, 2017.
- [5] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *ArXiv preprint arXiv:1409.0473*, 2014.
- [6] J. Weizenbaum, “Eliza - a computer program for the study of natural language communication between man and machine,” *Communications of the ACM*, vol. 9, no. 1, pp. 36–45, 1966.
- [7] K. M. Colby, “Ten criticisms of parry,” *ACM SIGART Bulletin*, no. 48, pp. 5–9, 1974.
- [8] D. A. Ferrucci, “Introduction to “this is watson”,” *IBM Journal of Research and Development*, vol. 56, no. 3.4, pp. 1–1, 2012.
- [9] I. Sutskever, O. Vinyals, and Q. V. Le, “Sequence to sequence learning with neural networks,” in *Advances in neural information processing systems*, 2014, pp. 3104–3112.
- [10] N. Kalchbrenner and P. Blunsom, “Recurrent continuous translation models,” in *EMNLP*, vol. 3, 2013, pp. 413–422.
- [11] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *ArXiv preprint arXiv:1406.1078*, pp. 1724–1734, 2014.
- [12] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio, “Show, attend and tell: Neural image caption generation with visual attention,” in *International Conference on Machine Learning*, 2015, pp. 2048–2057.
- [13] O. Vinyals, L. Kaiser, T. Koo, S. Petrov, I. Sutskever, and G. Hinton, “Grammar as a foreign language,” in *Advances in Neural Information Processing Systems*, 2015, pp. 2773–2781.

Bibliography

- [14] F. Zivkovic and D. Chen, “Trumpbot: Seq2seq with pointer sentinel model,”
- [15] N. Srivastava, E. Mansimov, and R. Salakhudinov, “Unsupervised learning of video representations using lstms,” in *Proceedings of the 32nd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, vol. 37, 2015, pp. 843–852.
- [16] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” *ICML (3)*, vol. 28, pp. 1310–1318, 2013.
- [17] S. Hochreiter, “Untersuchungen zu dynamischen neuronalen netzen,” Diploma Thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.
- [18] R. Pascanu, T. Mikolov, and Y. Bengio, “Understanding the exploding gradient problem,” *CoRR*, abs/1211.5063, 2012.
- [19] P. J. Werbos, “Backpropagation through time: What it does and how to do it,” 10, vol. 78, IEEE, 1990, pp. 1550–1560.
- [20] S. Hochreiter and J. Schmidhuber, “Lstm can solve hard time lag problems,” in *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, 1997, pp. 473–479.
- [21] A. Graves, “Generating sequences with recurrent neural networks,” *ArXiv preprint arXiv:1308.0850*, 2013.
- [22] H. T. Siegelmann and E. D. Sontag, “On the computational power of neural nets,” *Journal of computer and system sciences*, vol. 50, no. 1, pp. 132–150, 1995.
- [23] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, 2016.
- [24] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *ArXiv preprint arXiv:1412.3555*, 2014.
- [25] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, “Convolutional lstm network: A machine learning approach for precipitation nowcasting,” in *Advances in Neural Information Processing Systems*, 2015, pp. 802–810.
- [26] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” *ArXiv preprint arXiv:1509.00685*, 2015.
- [27] R. Desimone and J. Duncan, “Neural mechanisms of selective visual attention,” *Annual review of neuroscience*, vol. 18, no. 1, pp. 193–222, 1995.
- [28] L. Itti, C. Koch, and E. Niebur, “A model of saliency-based visual attention for rapid scene analysis,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 20, no. 11, pp. 1254–1259, 1998.
- [29] V. Mnih, N. Heess, A. Graves, *et al.*, “Recurrent models of visual attention,” in *Advances in neural information processing systems*, 2014, pp. 2204–2212.
- [30] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende, and D. Wierstra, “Draw: A recurrent neural network for image generation,” *ArXiv preprint arXiv:1502.04623*, 2015.

Bibliography

- [31] K. Cho, A. Courville, and Y. Bengio, “Describing multimedia content using attention-based encoder-decoder networks,” *IEEE Transactions on Multimedia*, vol. 17, no. 11, pp. 1875–1886, 2015.
- [32] D. von Grünigen, M. Weilenmann, J. Deriu, and M. Cieliebak, “Potential and limitations of cross-domain sentiment classification,” *SocialNLP 2017*, pp. 17–25, 2017.
- [33] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, *et al.*, “Tensorflow: Large-scale machine learning on heterogeneous distributed systems,” *ArXiv preprint arXiv:1603.04467*, 2016.
- [34] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky, *et al.*, “Theano: A python framework for fast computation of mathematical expressions,” *ArXiv preprint arXiv:1605.02688*, 2016.
- [35] R. Collobert, K. Kavukcuoglu, and C. Farabet, “Torch7: A matlab-like environment for machine learning,” in *BigLearn, NIPS Workshop*, 2011.
- [36] R. Rehurek and P. Sojka, “Software framework for topic modelling with large corpora,” in *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, May 2010, pp. 45–50.
- [37] A. A. Hagberg, D. A. Schult, and P. J. Swart, “Exploring network structure, dynamics, and function using NetworkX,” in *Proceedings of the 7th Python in Science Conference (SciPy2008)*, 2008, pp. 11–15.
- [38] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: Analyzing text with the natural language toolkit.* ”O’Reilly Media, Inc.”, 2009.
- [39] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, “The numpy array: A atructure for efficient numerical computation,” *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.
- [40] J. D. Hunter, “Matplotlib: A 2d graphics environment,” *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.
- [41] E. Jones, T. Oliphant, and P. Peterson, “Scipy: Open source scientific tools for python,” 2014.
- [42] B. Thirion, E. Duschenay, V. Michel, G. Varoquaux, O. Grisel, J. VanderPlas, A. Granfort, F. Pedregosa, A. Mueller, and G. Louppe, *Scikitlearn*, 2016.
- [43] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching word vectors with subword information,” *ArXiv preprint arXiv:1607.04606*, 2016.
- [44] S. Jean, K. Cho, R. Memisevic, and Y. Bengio, “On using very large target vocabulary for neural machine translation,” *CoRR*, vol. abs/1412.2007, 2014. [Online]. Available: <http://arxiv.org/abs/1412.2007>.
- [45] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.
- [46] H. Nguyen, D. Morales, and T. Chin, “A neural chatbot with personality,” [Online]. Available: <http://web.stanford.edu/class/cs224n/reports/2761115.pdf>.

Bibliography

- [47] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014.
- [48] J. Li, M. Galley, C. Brockett, J. Gao, and B. Dolan, “A diversity-promoting objective function for neural conversation models,” in *Proceedings of NAACL-HLT*, 2016, pp. 110–119.
- [49] S. Wiseman and A. M. Rush, “Sequence-to-sequence learning as beam-search optimization,” *ArXiv preprint arXiv:1606.02960*, 2016.
- [50] K. Yao, G. Zweig, and B. Peng, “Attention with intention for a neural network conversation model,” *CoRR*, vol. abs/1510.08565, 2015.
- [51] I. V. Serban, A. Sordoni, Y. Bengio, A. C. Courville, and J. Pineau, “Hierarchical neural network generative models for movie dialogues,” *CoRR*, vol. abs/1507.04808, 2015.
- [52] M. D. Zeiler, “Adadelta: An adaptive learning rate method,” *ArXiv preprint arXiv:1212.5701*, 2012.
- [53] S. Ruder, “An overview of gradient descent optimization algorithms,” *ArXiv preprint arXiv:1609.04747*, 2016.