ZHAW
Zurich University of
Applied Sciences

Bachelor-Thesis

FS 2017

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

Authors:
Dirk von Grünigen
Martin Weilenmann

Supervisors:
Dr. Mark Cieliebak
Dr. Stephan Neuhaus
Jan Deriu

June 1, 2017

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Zurich University
of Applied Sciences

School of Engineering

DECLARATION OF ORIGINALITY

Project Work at the School of Engineering

By submitting this project work, the undersigned student confirm that this work 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 project work 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 project works submitted.

Zusammenfassung

In der vorliegenden Arbeit werden die Möglichkeiten von Crossdomain Sentiment-Analyse mithilfe von Convolutional Neural Networks untersucht. Dabei ist das Ziel zu eruieren, inwiefern sich ein einzelner Sentiment-Klassifizierer auf verschiedenen Domänen verhält und ob es Sinn macht, Datensätze mehrerer Domänen in Kombination zu verwenden.

Als Erstes wird analysiert wie stark der Einfluss unterschiedlicher Word-Embeddings und Distant-Phasen auf einen Sentiment-Klassifizierer im Generellen ist und ob ein Zusammenhang zwischen den einzelnen Domänen und der besten Kombination an Word-Embeddings und Distant-Phasen existiert. Die Resultate zeigen, dass nicht anhand eines generellen Schemas entschieden werden kann welches die optimale Wahl ist. Dies muss im Einzelfall analysiert werden.

Im zweiten Teil wird untersucht, wie sich verschiedene Domänen im gegebenen Kontext verhalten. Dabei werden verschiedene Sentiment-Klassfizierer auf unterschiedlichen Domänen trainiert und dann auf allen anderen Domänen evaluiert. Die Resultate zeigen, dass es keine einzelnen Domänen gibt, welche sich besser eignet, um einen generalistischen Sentiment-Klassifizierer zu trainieren. Die Varianz der Resultate ist erstaunlich hoch, was nicht nur mit den unterschiedlichen Domänen, sondern auch mit den eingeführten Eigenschaften der Eindeutigkeit und Konzentration der Sentiments in Texten zusammenhängt. Die Generalisierung von einer Domäne zu einer anderen ist meistens nicht ohne weiteres möglich.

Im nächsten Teil wird untersucht, ob es sich lohnt, Daten aus unterschiedlichen Domänen für das Training eines Sentiment-Klassifizierers zu verwenden. Dafür werden sogenannte "Augmentation" Experimente durchgeführt: Bei diesen werden Daten aus mehreren Domänen in verschiedenen Verhältnissen durchmischt und danach wird der resultierende Sentiment-Klassifizierer auf einzelnen Domänen evaluiert. Die Resultate der Experimente zeigen, dass sich dieses Vorgehen von Vorteil ist, sofern zu wenige annotierte Daten der Zieldomäne zur Verfügung stehen.

Zuletzt wird untersucht, wie die Performanz eines generalistisch ausgelegten Sentiment-Klassifiziers auf einzelnen Domänen ist. Dafür wurden mehrere dieser Klassifizierer auf sogenannten "Ablation" Datensätzen trainiert. Dabei kann festgestellt werden, dass die Performanz eines generalistischen Sentiment-Klassifiziers auf einzelnen Domänen nicht bedeutend schlechter ist und für gewisse Anwendungsfälle durchaus eine Option darstellt.

Abstract

In the following work we are going to investigate the potential of crossdomain sentiment-classification using convolutional neural networks. The main point of this research is to explore the topic of combining data from multiple sources to train such sentiment-classifiers.

In the first part, we analyse how well the sentiment-classifiers for different domains work together with multiple combinations of distant-phases and word-embeddings. It is shown, that there's no a priori answer on which combination is the best upfront; it has to be evaluated on each domain separately.

The next part focuses on how well specialized sentiment-classifiers work on different domains. For this purpose, multiple sentiment-classifiers are trained on a single domain and evaluated on all others. The results show, that there is no single best domain to train a generalized classifier and that the performance deteriorate if different domains are used for training and testing. Further, we determined that ambiguity in sentiments and the length of texts has an impact on the performance of the resulting classifier.

We conducted augmentation experiments to investigate the issue of combining data from multiple domains to train a sentiment-classifier. This means that different combinations of data from multiple domains is used to train the classifiers. The results of the experiments show, such an approach is only favorable if there is too little annotated data of the target domain.

As the last point, we evaluate the generalization performance of a sentiment-classifier. For this purpose, we used ablation datasets, which are combinations of all domains except for the target domain to train the classifiers. The results show that the performance of a generalized sentiment-classifier is only slightly worse than the one of specialized classifers and can such a procedure can even be useful in certain use-cases.

Contents

Preface

Danke Stephan, Mark und Jan! :D

1. Introduction

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

2. Related Work

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

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 *conversation*. 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 the 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 [30], 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 on 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 us 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 them 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're 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 **??**) 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 behaviour across the temporal dimension of the input data".

Before explaining how this recurrence can be used to solve the weather prediction problem, let's 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 models 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.

For output and input m?

[1] http://r2rt.com/static/images/NH_VanillaRNNcell.png

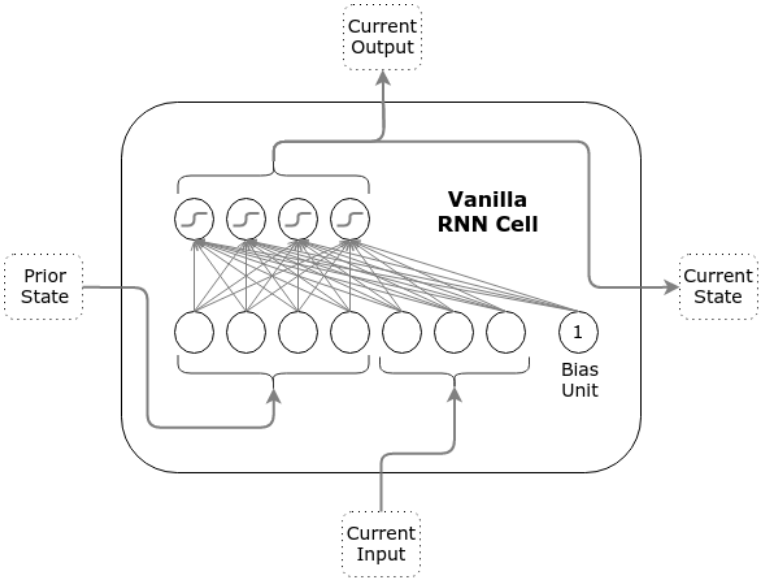


Figure 3.1.: Internal structure of a vanilla RNN.[1]

The internal structure is similar to the one of NNs, except for the recurrence which we are going to elaborate on below.

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

Format correctly!

In the equations above, one can see different variables with different indices which we'll 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 o_t are computed. The value of o_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 **??**, 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 behaviour 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 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 informations 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 informations on a daily basis, we would need to remember the informations 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 the problem of vanishing and exploding gradients explained in the next paragraph.

Vanishing / Exploding Gradient Problem 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 back-propagation by using gradient-descent in such a way, that the loss function is minimized w.r.t. to these parameters for the given training samples. This means, that we have to compute the derivates of the loss function by using the chain-rule $dz/x = (dz/dy)(dy/dx)$. The entirety of all these derivates w.r.t. to the parameters is called the gradient. Since the gradient also flows through the activation functions φ_h of the recurrence, this computations also include the derivates 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 derivates possibly leads to an exploding or vanishing values. This has a severe effect on training the RNN: If the value of the derivates is exploding, this means that the weights are adjusted in a way which we would call "hyperbolism" and this leads to a network where the predictions get worse and worse. On the other hand, 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 with large window sizes. Rework

The problem of vanishing gradients was first analyzed by Hochreiter in his diploma thesis [16]. 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 [26]. 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 [35].

Long Short-Term Memory Networks To solve the problem with vanishing/exploding gradients mentioned before, Schmidhuber and Hochreiter introduced an advanced version of a RNN cell, called *Long Short-Term Memory*, or LSTM in short [17]. The structure of this kind of cells is much more sophisticated then vanilla RNN cells, but it also solves the two most pressing problems: Vanishing/exploding 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 informations 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.

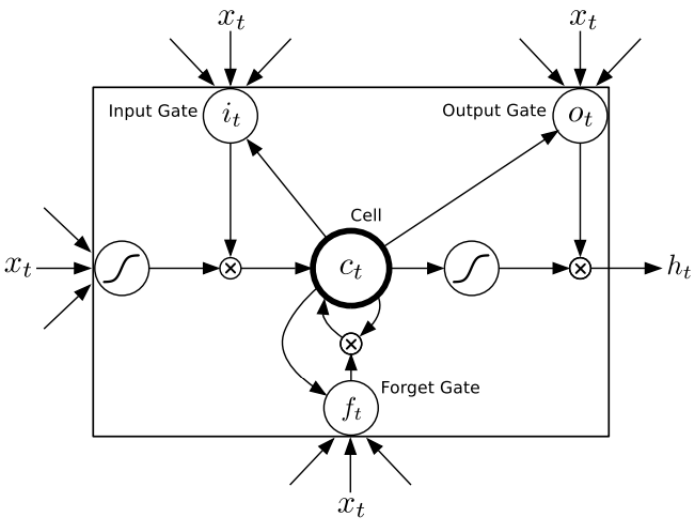


Figure 3.2.: Internal structure of a LSTM cell [11].

[1] http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/

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 [29]. However, in practice this structure solves all of the aforementioned problems to at least some degree.

First, let us look into the vanishing/exploding gradient problem: The problem occured 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 (if the derivate is below 1) or exploding (if the derivate is above 1) gradients. The LSTM cell solves this problem by not applying an activation function on the recurrence, but rather using the identity function $f(x) = x$ whose derivation is always 1. This practically solves this problem because the gradients can neither explode nor vanish anymore.

The second problem is that an RNN cell has no easy way of forgetting information stored in the hidden state. This is obviously solved by the introduced 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 a empirical study by Greff et al. [12]. There are even more architectures for RNN cells like *Gated Recurrent Unit* [7] and *Convolutional LSTM* [36], which we will not go into detail on in this thesis.

3.3. Sequence-To-Sequence Learning

The RNN and LSTM cells described in the preceeding 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 [30][21][6], 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 at a time. 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 t given the hidden state from the decoder at time step $t-1$ and the last output y_{t-1}.

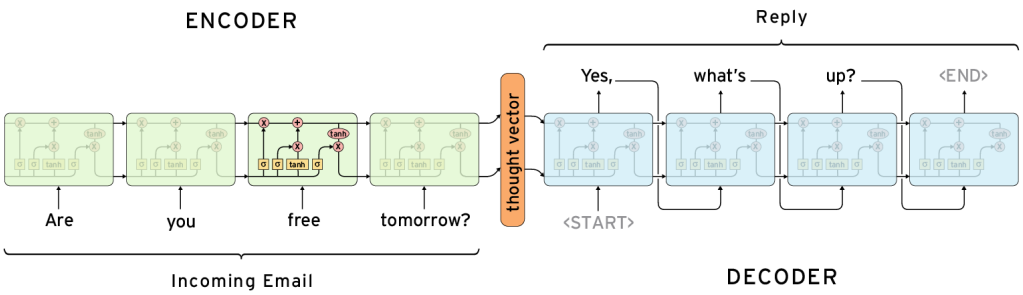


Figure 3.3.: Internal structure of a Sequence-To-Sequence Model.[2]

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're talking about language modeling, the model is conditioned to optimize the conditional probability $p(y_1, \ldots, y_T | x_1, \ldots, x_{T'})$, 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 $x_1, \ldots, x_2$, and passes this to the decoder for generating the output sequence. The decoder is then trained to compute the probability of $y_1, \ldots, y_T$ given the thought vector v:

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

Describe more?

[2] http://suriyadeepan.github.io/2016-12-31-practical-seq2seq/

Each of the terms $p(y_t|v, y_1, \ldots, y_{t-1})$ in the product represents a distribution over all words of the vocabulary that is computed when applying the softmax layer at the end.

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 $x_1, \ldots, x_T$ 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 $y_1, \ldots, y_N$ 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 $y_1, ..., y_{T'}$ 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. The generality of this model has been shown several times !!REFERENZEN!!.

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* [2] 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.

revisit one more time!

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 [32] to do so. Formally, the attention mechanism works as follows: Consider that we already have all hidden states of the encoder $\mathbf{H} = \{h_1, \ldots, h_t\}$ 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 $\mathbf{D} = \{d_1, \ldots, d_t\}$ (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 as almost anywhere else. 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.

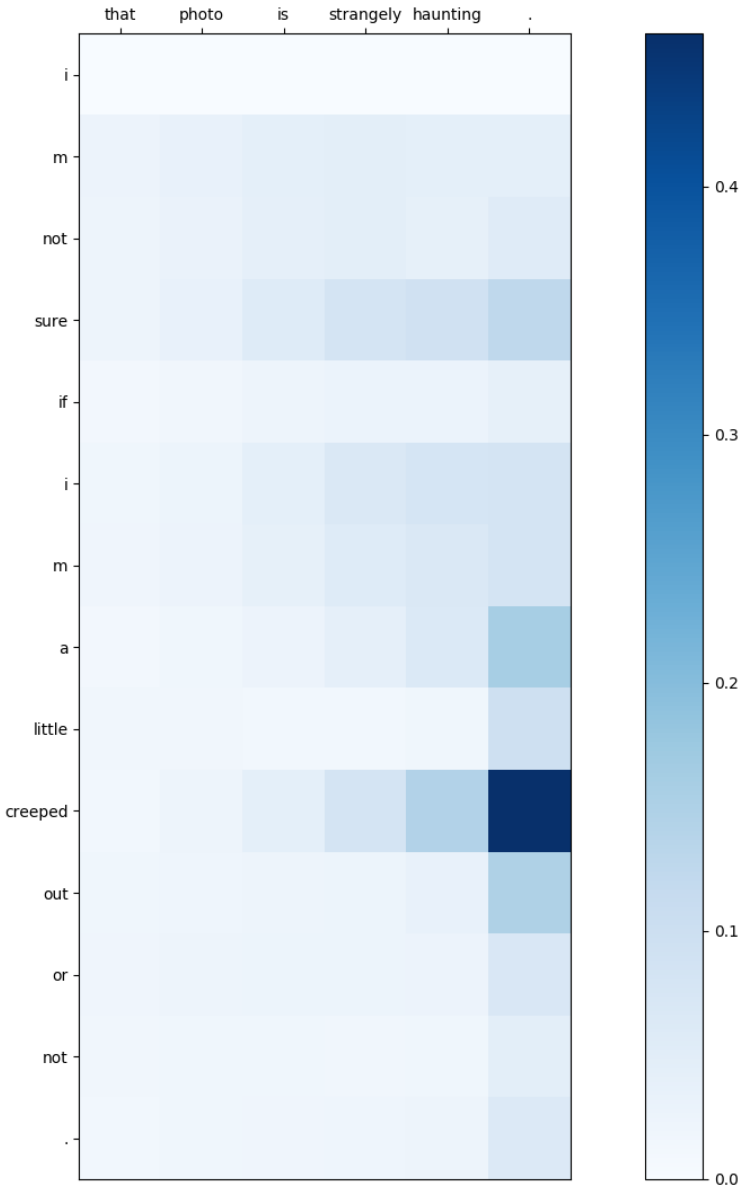


Figure 3.4.: Example on how attention can be visualized using a heatmap. The input sequence is on the x-axis and the respective response on the y-axis. Each line in the heatmap can be interpreted as the probability distribution over the thought vectors of the encoder which the decoder accesses at each 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 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 [9][19][23] and has a wide range of applications there [13][37][5]. 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 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 word y_t to produce the output token 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 last hidden state (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 probability. 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 seconds 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 has to be 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 atmost 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-probabilites 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, the This process is then carried out until we reach an `EOS` token in one the beams or the maximum number of allowed time steps in the decoder is reached. To decide, which is the best predicted sequence at the end of the beam-search, we again look at the sum of the log-probabilites and take the one with the value which is nearest to 0 (as a sum of log-probabilities is a sum of negative numbers).

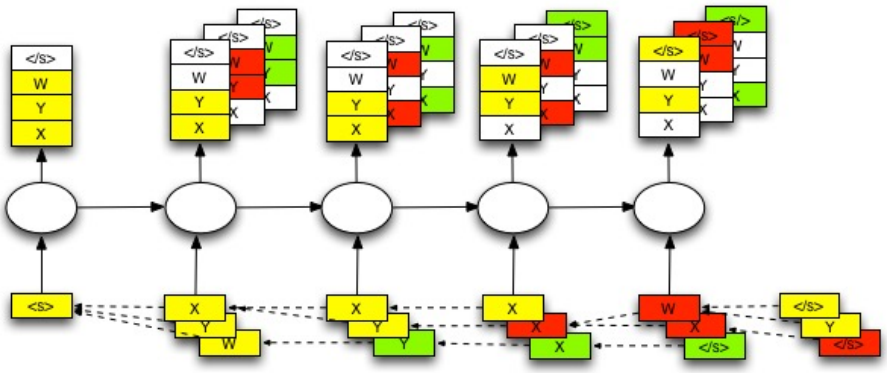


Figure 3.5.: Visualization on how beam search works in the context of seq2seq models.[3]

3.4. Performance Metrics

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

Cross-Entropy Loss For our model, we are using the *Cross-Entropy* loss function while training. 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 whole training dataset:

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

Check one more time!

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. The perplexity can be computed by the following equation:

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

The perplexity tells "how good" a language model is. More formally, it tells us how good the model reproduces the data 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 $H(X)$. This means, we effectively have one performance metric to use. To fix this problem, we also investigated into using Sent2Vec[4] [25] for measuring the semantic difference of the sentences generated by the model when testing it after it has been trained. We propose the following way to test the model:

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 model 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 this 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 experiments 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 are going to describe the environment in which this software system can be used and what kind of external software and hardware was used to run the experiments 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 experiments of this thesis. The following properties were identified which the developed system should fulfill:

- **Parameterizable**: All experiments should be allowed to run in parameterized manner, which means that all important hyperparameters, training data and places to store the different results should be parameterizable through a JSON[1] configuration file.

- **Reproducible**: It should be possible to run experiments multiple times without having to put additional efforts into it.

- **Analyzable**: This means that all results from all experiments 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 experiments were going to be 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 an experiment. 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 training data the model was already trained on as well as loading all the metric 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 possiblity 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 experiments without the fear of loose 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 thought this system was used to perform sentiment-analysis with convolutional neural networks [14]. 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 [1][31][8].

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.

On-going 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 framework[4] specifically for seq2seq models with `TensorFlow`. Of course we were excited to try it out and so we started with conducting experiments using the framework some days later. At first, everything looked fine and the training seemed to be working by the look of it. 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 bug[5] that also troubled as 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 situtation 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 an experiment. 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

[4]https://github.com/google/seq2seq
[5]https://github.com/google/seq2seq/issues/103

in a tutorial of the `TensorFlow` team on how to implement a simple neural machine translation system using their framework[6]. This implementation however 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 this 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 [33], 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 whole softmax weights matrix to do 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 [33]), 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` version[11]. 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 the by fast the most interesting part, as we had the possibility to work on interal 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 experiments.

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 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

[7] https://github.com/google/seq2seq/blob/master/seq2seq/inference/beam_search.py
[8] https://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
[11] https://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 GitHub[12]. 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 `flask`[13] for implementing the backend system and a combination of `jQuery`[14] JavaScript library and the `boostrap`[15] frontend framework.

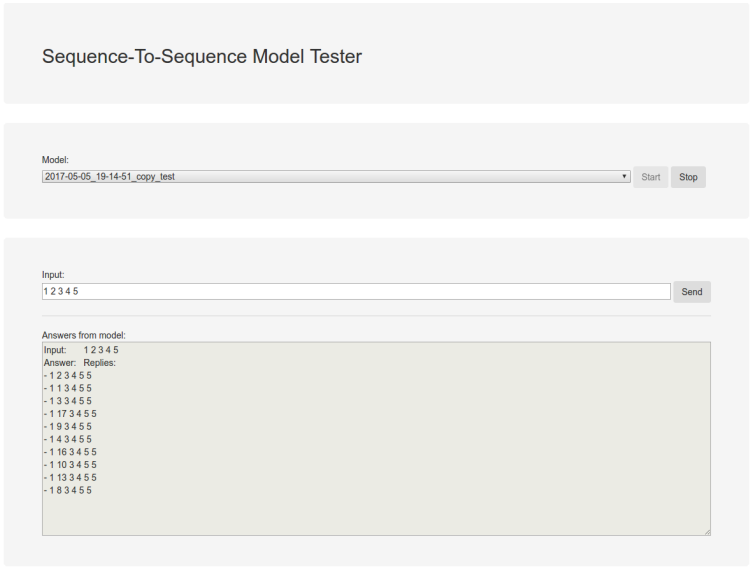


Figure 4.1.: Frontend showing the output when sending the sequence "1 2 3 4 5" to a model trained on the copy task (see chapter 4.3).

4.5. Scripts

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

[12] https://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 data 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.
- Scripts for generating and managing experiments.

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

4.6. Hardware

All experiments have been conducted on the GPU-cluster the InIT (Institut für angewandte Informationstechnologie) has provided us for running the experiments related to our thesis. On this server, there are 8 Nvidia Titan X (Pascal) GPUs are installed and a total of 24 CPUs with 501GB of RAM stood at our disposal. The experiments 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 experiments 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 an went on with the biggest possible model which fits on a single GPU (see chapter !!REFERENZ!!).

4.7. Operating System & Software Packages

As mentioned before, all experiments 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. 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	[27]
`h5py`	2.6.0	Website[22]
`matplotlib`	2.0.0	[18]
`networkx`	1.11	[15]
`nltk`	3.2.2	[4]
`numpy`	1.11.3	[34]
`TensorFlow`	1.0.1	[1]

Table 4.1.: Python libraries used in the system.

5. Data

In this chapter we are going to 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 corpora is also prepared in for later usage in the Chapter **??**.

5.1. Datasets

We are using two different datasets for the experiments. The first is the current version of the OpenSubtitles corpus [22] that contains subtitles from over 300′000 movies. The second is a datasets containing all comments posted on the website Reddit[1] from the year 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	[22]
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 datasets in several aspects: Since the OpenSubtitles corpus is extracted from subtitles of a large number of movies, it represents a corpora of *spoken* language. In contrast, the Reddit dataset only consists of comments posted on the website and hence represents written, partially colloquial, language. This two types of language usually have a a big discrepancy in how they are used[3].

5.2. Structure of the Raw Corpora

The following two paragraphs explain how the raw datasets are structured and how the raw utterances are extracted from them.

[1] https://www.reddit.com/

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

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

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.

```
<?xml version="1.0" encoding="utf-8"?>
<document id="4955664">
  <s id="1">
    <time id="T19S" value="00:01:54,300" />
    <w id="15.1">Here</w>
    <w id="15.2">it</w>
    <w id="15.3">'s</w>
    <w id="15.4">your</w>
    <w id="15.5">brain</w>
    <w id="15.6">versus</w>
    <w id="15.7">the</w>
    <w id="15.8">opponent</w>
    <w id="15.9">'s</w>
    <w id="15.10">.</w>
    <time id="T19E" value="00:01:58,250" />
  </s>
  ...
</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 from ranging from 2006 up 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 **??**) 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 childs of this root node 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).

```
{
  "retrieved_on": 1425820157,
  "parent_id": "t1_c02s9rv",
  "distinguished": null,
  "created_utc": "1199145604",
  "score_hidden": false,
  "subreddit": "reddit.com",
  "score": 4,
  "author_flair_text": null,
  "author_flair_css_class": null,
  "body": "Wow, you're a buzz-kill.",
  "ups": 4,
  "id": "c02s9s6",
  "archived": true,
  "downs": 0,
  "edited": false,
  "subreddit_id": "t5_6",
  "controversiality": 0,
  "name": "t1_c02s9s6",
  "link_id": "t3_648oh",
  "gilded": 0,
  "author": "Haven"
}
```

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

[4]https://docs.python.org/3.4/library/shelve.html

5.3. Preprocessing

The preprocessing process is explained in the following paragraphs. 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 on 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 the 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 wether 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 childs; in the tree these are A, B and C 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) for the root comment A. 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

Begrün
einfüge

corpus, 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.

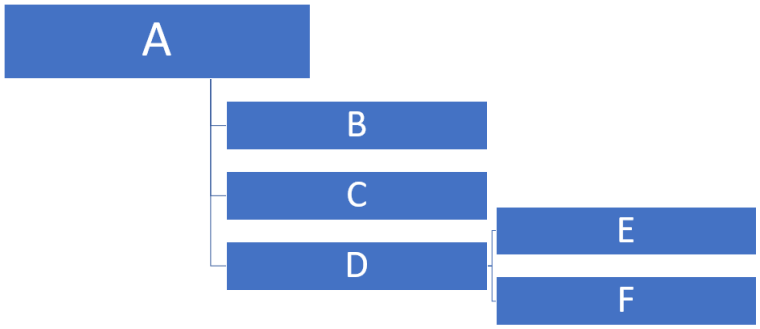


Figure 5.4.: Exemplary tree how the comment tree is structured for a single thread. A stands for the threads root node and B to F are comments.

5.4. Vocabulary and Coverage

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

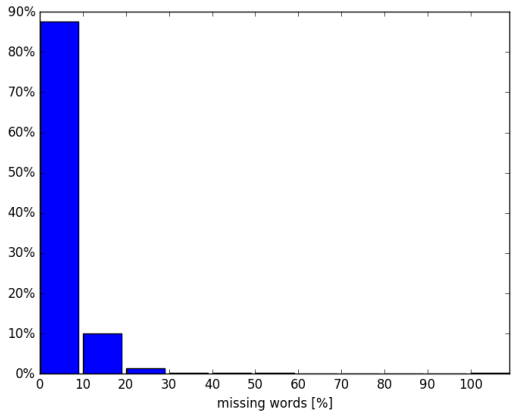
tabelle word coverage ist nicht referenziert?

	Size [Thousand]	No. of Words [Thousand]	No. of known Words [Thousand]	Perc. of known Words [%]	No. of unknown Words [Thousand]	Perc. 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%

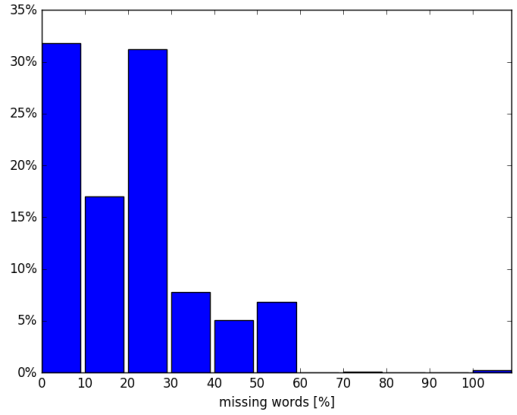
Table 5.2.: Word coverage of differently sized vocabularies extracted from the generated datasets.

Below, in figure 5.5, you can find plot where it shows how much 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.

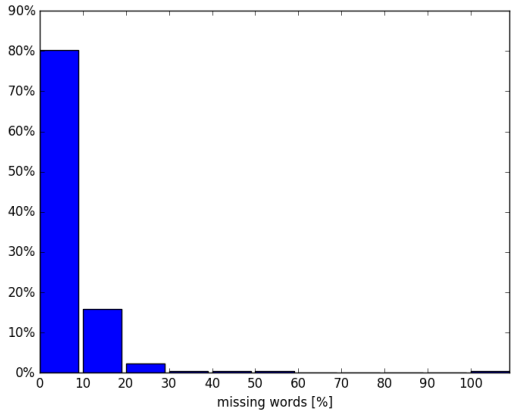
Begründe es befinden sich viele Sätze mehrfach im Datensatz.



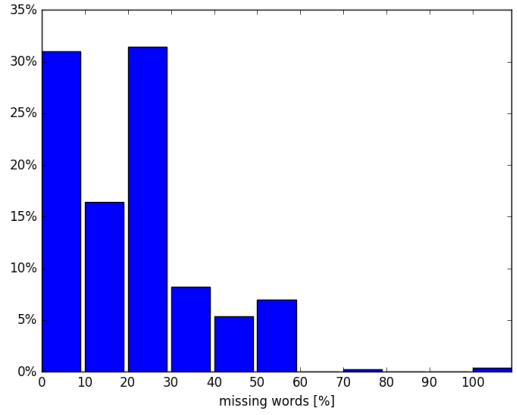
Reddit 100k



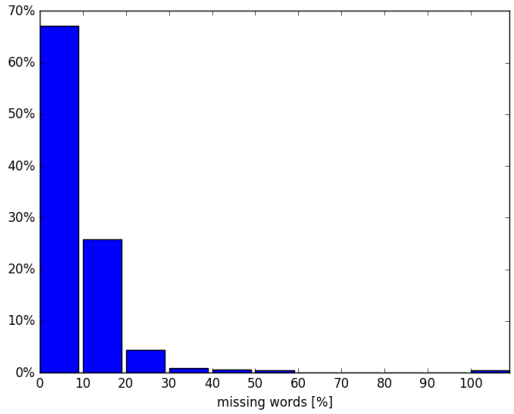
OpenSubtitles 100k



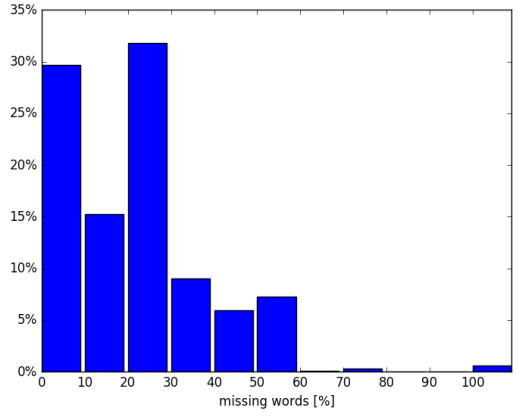
Reddit 50k



OpenSubtitles 50k



Reddit 25k



OpenSubtitles 25k

Figure 5.5.: Plots showing the percentage of words missing per utterance for specific vocabulary sizes.

Other blue color and replace "missing words" with "Miss-

5.5. Splitting the Datasets

The generated datasets have to be split in order to obtain a train, validation and test set. The table 5.3 shows how 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.

	Set	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 set.

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 there, where the timestamps signify a large time difference between the first and second utterance. This is comprehensible, as all utterances are just saved in the corpus one after each other without taking the 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 a large portion of the utterances have a time-lag higher than a certain timespan. The results of this investigation are summarized in the figure 5.6.

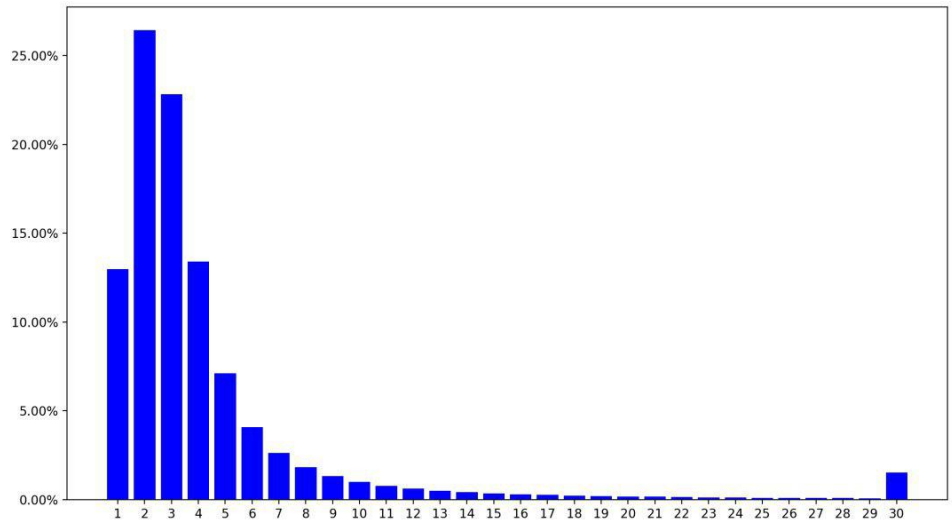


Figure 5.6.: Relative distribution (discrete) of the time-lag between two utterances in the raw OpenSubtitles corpus. Most of the utterances lie in the range from 1 to 5 seconds: 13.0% within 1 second, 26.4% within 2 seconds, 22.8% within 3 seconds, 13.4% within 4 seconds and 7.0% within 5 seconds.

As one can see, most of the utterances, actually over 80%, occur between 1 and 5 seconds after each other. The big spike at the end of the graph contains a much large portion than all the other between 10 and 30 seconds. This has to do with the fact, that there might be changes in the scenes of the movies, which can lead to two consecutive utterances in the corpus being uttered with a larger time 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.

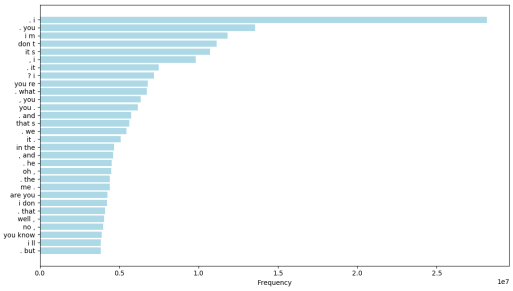
big? oder einfach nur spike?

As this analysis has shown, most of the utterances lie so close to each other, that this will most probably not be a problem when training our model with it.

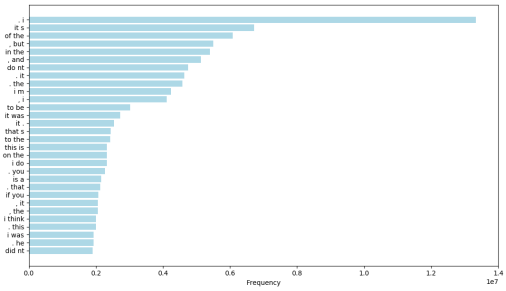
5.7. N-Gram Analysis

We also do an n-gram analysis of the used corpora. For this purpose, we generated uni- and bigrams for both the OpenSubtitles and Reddit corpora. We do this, to be able to compare the n-grams of the outputs produced by our models after they have been trained with the n-grams obtained when analyzing the preprocessed corpora. 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 results of analyzing the preprocessed datasets are visualized in the figures 5.7 and 5.8. For the second visualizations, we use a custom graph visualization which we call “N-Gram Graph”. This allows us to visualize how n-

grams can be connected to each other (through edges) and at the same time visualize their relative occurrence frequencies to all other n-grams in the graph.

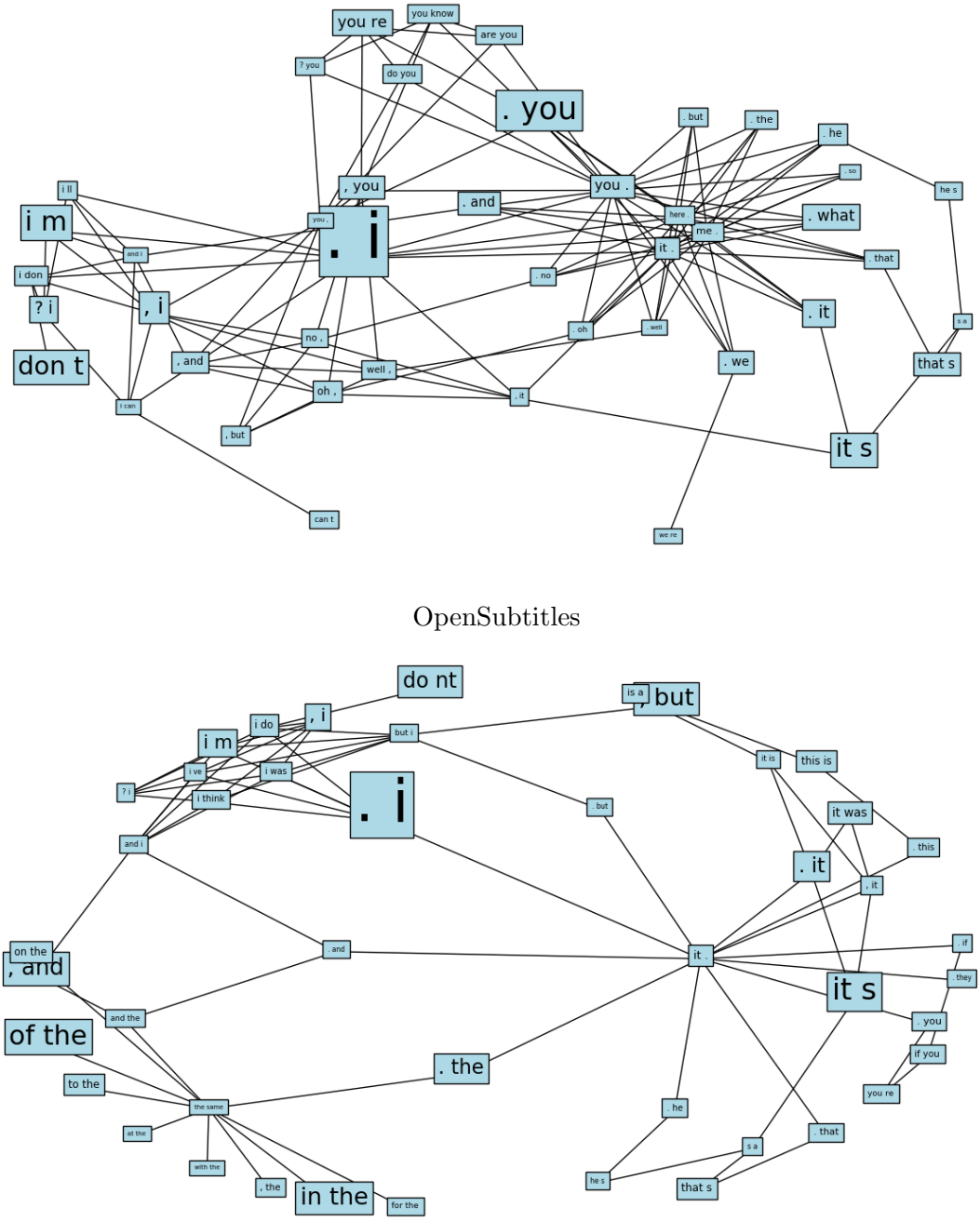


OpenSubtitles



Reddit

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



OpenSubtitles

Reddit

Figure 5.8.: N-Gram graphs for the 50 most used bigrams in the OpenSubtitles (upper) and Reddit (lower) datasets. Each node in the graphs represents a bigram, the edges between them show that either the first word of the first bigram matches the second word of the other bigram or the last word of the first bigram equals the last word of the second bigram. The size of each node is relative to its occurrence frequency, which means, that larger nodes occur more frequent than smaller ones.

6. Methods

In the following chapter, we are going to elaborate on how we performed the experiments, which hyperparameters where 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 are going to describe the architecture of the model used to conduct our experiments.

Sequence-To-Sequence In general, we are using the architecture of seq2seq models describe in chapter 3.3 with LSTM cells. We restrict ourselves to only use one LSTM cell for the encoder, and another cell for the decoder. This has to do with the fact, that we wanted our cells to be as large as possible to come as close to the size of the cells used in [33]. However, this is not simply possible due to the fact, that in the referenced paper 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 trained their models on a large number of CPUs due to this fact, because such a huge network fits does not fit in the memory of any GPU currently available. As we are seeking to train our models on a single GPU (see chapter 4.2), we had to shrink the size of our model to the biggest size possible so that it still fits within the 12GB of memory the GPUs we are using have (see chapter 4.6). The exact size of the model used in this thesis is described in the subsequent paragraph "Hyperparameters" below.

Down-Projection of Hidden State Because of the problematic with such large RNN cells as described in the preceding paragraph, we implemented a so-called *down-projection* at the end of the decoder cell. This is done similar to the down-projection used in [33], with the main difference lying in the motivation why we implemented it. In the paper, they state, that they used it 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 be able to without the projection. The implementation of this feature allowed us to grow our model in size by a factor of two, from a hidden state size of 1024 to 2048, without the need to

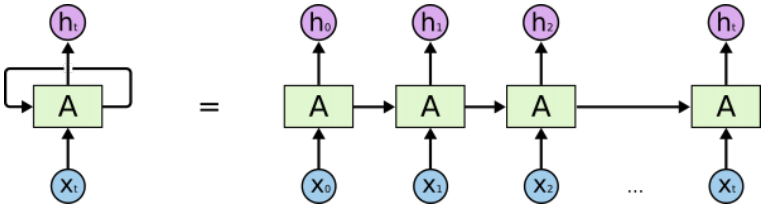


Figure 6.1.: Image for illustrating the process of unrolling an RNN over a fixed size of time steps.[2]

sacrifice the size of the vocabulary used (see chapter 6.3 for more information on the used hyperparameters).

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 [20] which is only applied while training the models. Basically, the idea behind it is, that instead of using the full softmax at each time step in the decoder, we only use a subset of the words which is randomly sampled (besides the words which is the target word) from the vocabulary to approximate the softmax layer. This speeds up training dramatically due to the fact, that not the whole softmax layer has to be used and hence not all derivates 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 Due to the fact, that 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 of the unrolled model (as each layer basically represents the same cell). With the latest `TensorFlow` API[1], 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 used 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 timespan we pick, 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 the need to shrink any other parts of the model.

[1] https://www.tensorflow.org/api_docs/python/tf/nn/dynamic_rnn
[2] http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Reversing the Input Sequence As [30] 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 [17] 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-system. Nevertheless, we want to try to use it and see if it makes any difference our models. We decided that the Reddit model should be the one that is trained with reversed input sequences.

wenn wir in den Resultaten OpenSubtitle reversed angeben dann müssen wir das hier präzisie

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 corpus. The first is trained on the OpenSubtitles corpus and the second on the Reddit corpus. Both of them are going to be trained in the same way for the same time, 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

Table 6.1.: Hyperparameters used for our seq2seq models.

Optimizer As the optimizer, we use *AdaGrad* [10] as in [33] with the learning rate set to 0.01. We also use gradient clipping and set the maximum allowed gradient value to be 10, as described in [26].

Training We train our models on the hardware described in chapter 4.6 with the software packages from chapter 4.7. As this models take a really long time to converge (in [33] the authors trained their model for several weeks), we also have to train ours for a long time. To show the development of the model throughout this time, we took several snapshots, approximately every third day or about after every 500'000 batches. The taken snapshots are going to be used in the analysis we are going to do in the next chapter. They enable us to not only do an analysis of the final results, but rather show the evolution of the models throughout the training process. The two models were trained over a time period of 20 days, each trained on 3 million batches, each containing 64 samples.

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 corpus. 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 analyse the difference between using the approaches of using a single big corpus, which we are not able to completely process, against a smaller corpus that we can iterate more than twice in the same time.

6.4. 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 **??**. 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 corpora. This also includes an analysis of the syntax and 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 other papers (e.g. [33]) and chatbots (such as CleverBot[3]).

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 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

[3]http://www.cleverbot.com/

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 to be alike the reference output. This thought is supported by results that Nguyen et al. [24] 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 metric, we are going to use Sent2Vec [25], 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 repository[4], as it is possible to embed new sequences without the requirement of retraining the models.

[4]https://github.com/epfml/sent2vec

7. Analysis of Results

In this chapter, we are going to analyze the evolution and final results of the two models (see chapter 6.2), trained on the OpenSubtitles and Reddit corpus, under several different aspects. This includes the analysis of the learning process throughout the training. Then we are going to analyse the language model of the trained models and try to find a correlation between the language model in the training dataset and the language model the trained models produce.In the end, we are going to focus on how well the model understands and percepts them, combined with an analysis

1. **Quantitive**: This part focuses on evaluating metrics and other measurements. We are going to look into the metrics at train time, a similarity analysis using Sent2Vec and an analysis of the produced language models by using n-gram collocations.
2. **Qualitative**: This part focuses on the quality of the language produced by the models. This includes a comparison between our models and *CleverBot*[1] and the development of the language model of the models throughout the training time.
3. **Miscellaneous**: This part focuses on theoretical analysis, such as analyzing the internal embeddings for the input sequence or if and how the models are using the soft-attention mechanism in their favor.

The analysis is done in the order listed above, that means we start by analysing the quantitive measures Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

* Attention * Learning process * Generated outputs over time * Generation at the end and comparison with cleverbot and paper * Generated outputs at the end and related ngram analysis * Generated embeddings for input sentences * Sent2Vec analysis

[1] http://www.cleverbot.com/

Diese Seite habe ich noch nicht genau angesch aber ich denke wir geben hier auch unsere grobe struk- tur bekannt Satz/Te set, n- Gramm Wörter, thought Vektor. Von Ganz aussen nach innen, sozusag

Two times model is ugly!

Not correct: What are we going to do with

7.1. How Did The Training Go?

First, let us start by analysing the evolution of the models with regard to the available performance metrics throughout the time span of the training. Below, in Figures 7.1 and 7.2, one can see the development of the loss and perplexity on the training sets for the two different models.

vielleicht müssen wir den anpassen wegen neuer struktur

OpenSubtitles What is eye-catching when comparing the two is that the OpenSubtitles seems to have much more variance in its performance on the validation set as opposed to the Reddit model. This probably comes from the fact that the OpenSubtitles datasets are much more noisey than the one of the Reddit model, as [33] also already noticed. This has to do with the fact, that we do not have any information about turn taking, which means it is certainly possible that consecutive utterances may be uttered by the same person even though we treat it if it were two persons. Also, there is the problematic with the time lags between utterances as analysed 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.

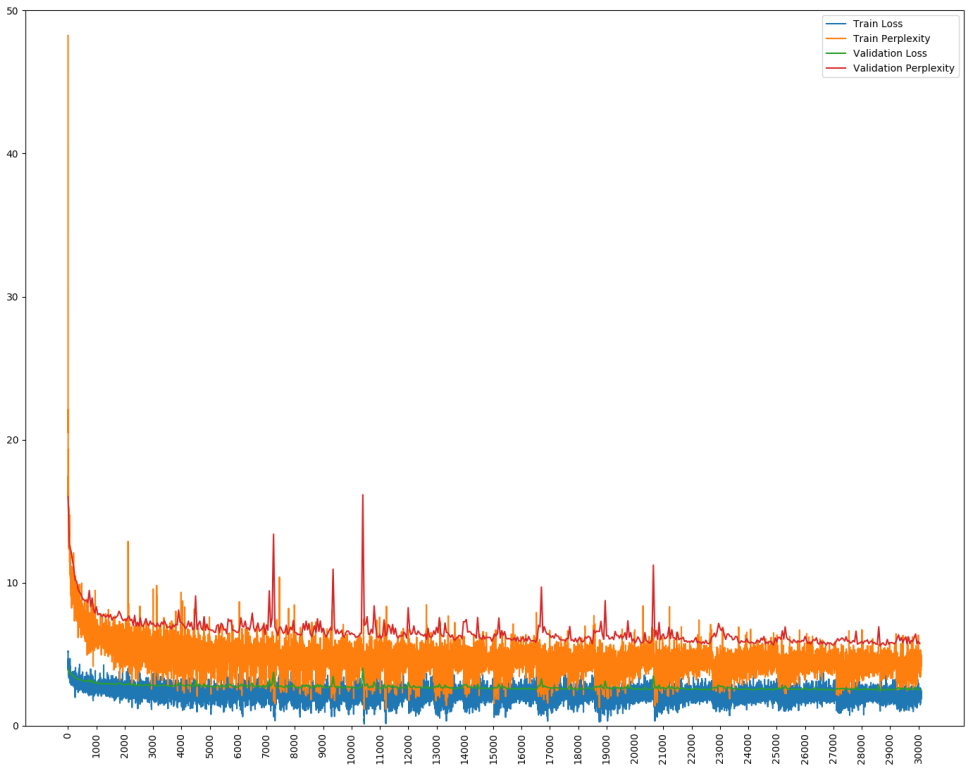


Figure 7.1.: Development of the loss and perplexity on the training and validation set throughout the training of the OpenSubtitles model. One tick on the x-axis is equal to 100 batches processed.

Einheit angeben (x = 1000-er)

Reddit The learning process of the Reddit model 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 training metrics. We cannot explain this behaviour currently. We assume that this comes from the fact...The variance however is much smaller than with the Reddit model, which strengthens our argument, that a well-structured dataset helps a lot when training such systems as it confuses the model much less.

Maybe we should be able?!

maybe rewrite this sentence somehow

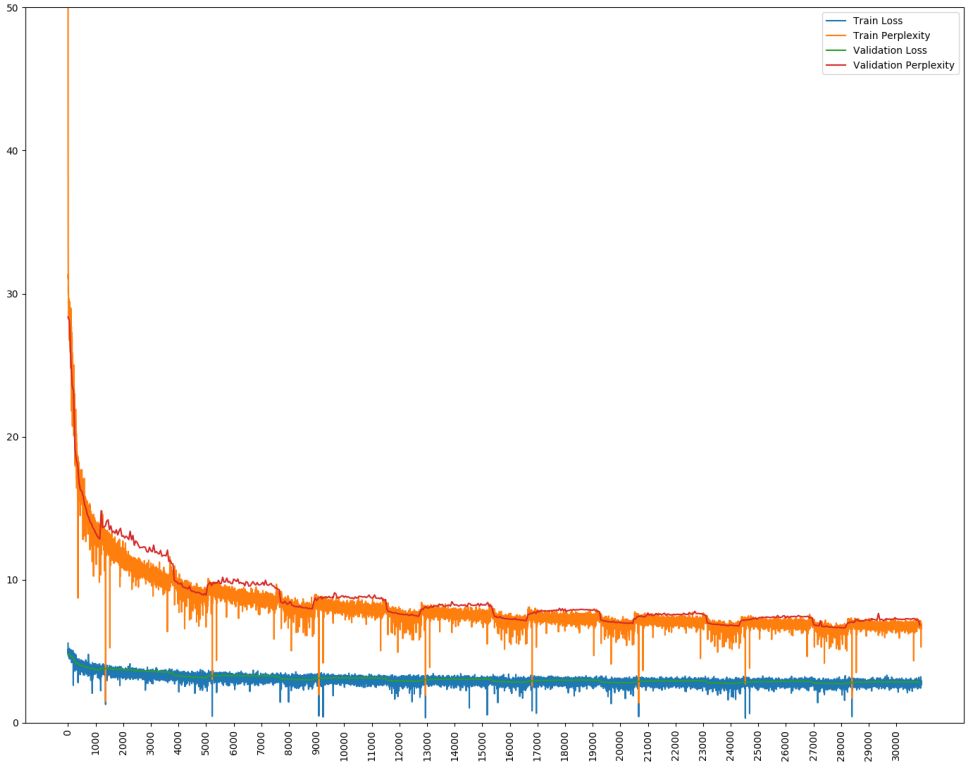


Figure 7.2.: Development of the loss and perplexity on the training and validation set throughout the training of the Reddit model. One tick on the x-axis is equal to 100 batches processed.

Einheit angeben (x = 1000-er)

Compared to each other In der Abbildung ?? sehen wir die loss Werte auf den Validierungsdaten beider Modell in einem Bild. Hier das Bild mit der valid los/perplex beider Modelle referenzieren und kurz schreiben, wer wie schnell gelernt hat und wie sich die Kurve unterscheiden.

können wir auch weglassen diesen paragraphen Weiss nicht welche Erkenntnis wir daraus ziehen können

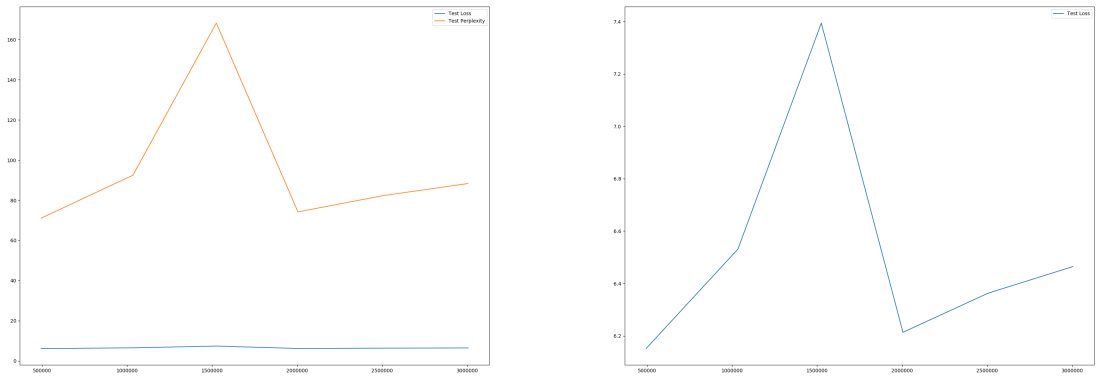
The Training Seems Successful From the looks of the plots, it looks like the training went fine for both models, as both of them have degrading loss and perplexity. We also saw differences in how the models have evolved over the time span of the training, especially the dips in the Reddit model, but we cannot conclude anything from this right now. We are going to analyze this further and see if we can find any reasons for it. 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 asses the performance of these models on our test datasets. Here we use the same metrics as within the training, namely the cross-entropy loss and perplexity values. We have evaluated each model on the respective test dataset for each checkpoint we have created while training (see Chapter 6).

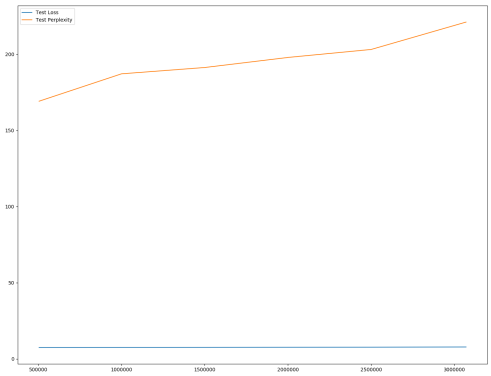
Surprising Results The results on the test set are quite the opposite of the results from the training process (see Figures 7.3 and 7.4), where both of the models are getting worse over time. We did not expect that, but nevertheless, we are going to analyze the problem. The results of the OpenSubtitles model seem to vary across the different checkpoints, with the best result having a perplexity of 71.07 and a loss of 6.15 and coming from the evaluation with the first checkpoint. The best result of the Reddit model is also achieved on the first checkpoint, with all other checkpoints having a worse perplexity. This result stands in contradiction to what we have expected. Instead of the loss going up, we would have expected it to go down in the same way as it did on the training and validation datasets. We assume, that this has to do with 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 immensely high. For this reason, we proposed a third performance metric, namely the usage of Sent2Vec [25] embeddings, to measure the similarity between the expected and generated responses. Before we are going to do this analysis, we want to take a look at different samples from both models to show that they indeed improved over time, even thought the test metrics tell a different story.

Maybe some more clarification on what analyse means?

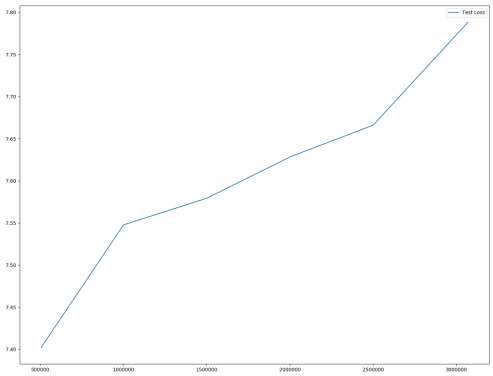


The loss and perplexity. Only the loss.

Figure 7.3.: The loss and perplexity by running the six different snapshots against the test dataset using the OpenSubtitles model.



The loss and perplexity.



Only the loss.

Figure 7.4.: The loss and perplexity by running the six different snapshots against the test dataset using the Reddit model.

Progress Is Achieved As the results of evaluating the models on the test dataset has shown, it seems that the models did not make an meaningful progress throughout the training. For this reason, we decided that we would like to evaluate the model by hand and see if the generated responses improve when using the different six snapshots of the models. We opted for four different input sentences, that means two for each model, and generated responses for each of the snapshots. The results can be seen in the Table 7.1 for the Reddit model and Table 7.2 for the OpenSubtitles model.

fuck title

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 7.1.: Two example dialogs with the Reddit model with 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`

Format so that all look the same, probably by using tabularx and an X column

Table 7.2.: Two example dialogs with the OpenSubtitles model with one response per snapshot.

As seen in the examples above, there has indeed been an improvement in the answers, what stands in contradiction to the development of the performance of the models on the test datasets. Our conclusion from this is, again, that the cross-entropy loss and perplexity are not fit to asses if the responses are meaningful.As we have already described in Chapter 3.4, we were aware of the fact, that our current set of metrics is not satisfying for evaluating if the responses are meaningful, why we decided to use yet another metric, namely Sent2Vec embeddings, which we are going to use in the next section.

Write more?

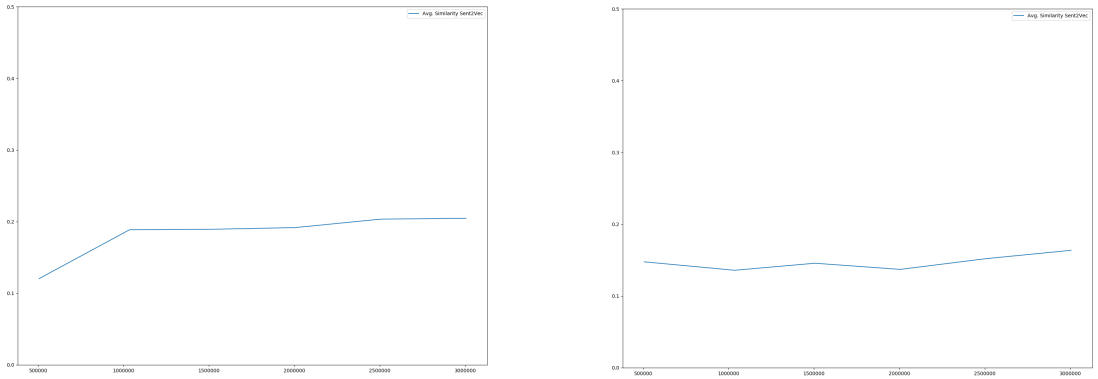
Sent2Vec Analysis As described in Chapter 3.4, we use Sent2Vec embeddings for measuring the semantic similarity between the generated and expected responses on the test dataset. For this purpose, we have used the pretrained models available on the GitHub page of the project[2]. We have decided, that we use both the *Twitter* and *Wikipedia* models for the assessment. The results can be seen in the Figures 7.5 and 7.6. What can be directly seen is that both the Reddit and OpenSubtitles models perform better on the pretrained Wikipedia model in comparison to the pretrained twitter model. This probably has to do with the fact that "compressed" language used when writing tweets (e.g. "w/o" instead of "without") and with the coverage of the vocabulary, which is higher with the Wikipedia model.However, both results are pretty bad, with the Reddit

Need to check that!

[2]https://github.com/epfml/sent2vec

results being much better than the OpenSubtitles, about twice as good (see Tables 7.3 and 7.4). The OpenSubtitles model starts with an average similarity of 0.167 for the first snapshot and climbs up to 0.204 for the last snapshot. The Reddit model starts with an average value of 0.336 for the first snapshot and increases to 0.359 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. In summary, the results of both models are pretty bad, as the maximum similarity is 1.0.

Write, or remove that?

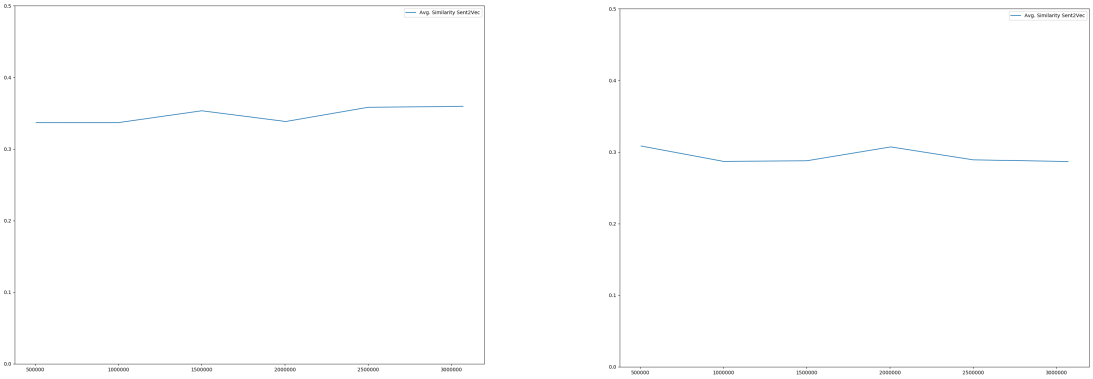


Wikipedia Twitter

Figure 7.5.: Results of the evaluation with Sent2Vec on the outputs of the OpenSubtitles models using the pretrained models. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity when using Sent2Vec for each snapshot.

Snapshot	Avg. Similarity (Wikipedia)	Avg. Similarity (Twitter)
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 7.3.: The average similarities when applying the Sent2Vec metric on the expected and generated responses from the OpenSubtitles model.



Wikipedia Twitter

Figure 7.6.: Results of the evaluation with Sent2Vec on the outputs of the Reddit models using the pretrained models. The ticks on the x-axis show the different snapshots and the y-axis the average semantic similarity when using Sent2Vec for each snapshot.

Snapshot	Avg. Similarity (Wikipedia)	Avg. Similarity (Twitter)
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 7.4.: The average similarities when applying the Sent2Vec metric on the expected and generated responses from the Reddit model.

Fazit for this paragraph?

Generic Response are a Problem One potential reason for the bad results when using the Sent2Vec metric is that we witnessed both models generating generic responses a lot of the time. To analyze this our first idea was 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 the models predict a lot of time (see Tables 7.5 and 7.6).

Sentence	Frequency
i m not gon na let you go	41853
i m not sure i can trust you	21263
i m not gon na say anything	9163
i m not gon na let that happen	7426
i m sorry	7235
you re not gon na believe this	7068
you re not gon na believe me	6878
i m not gon na hurt you	4829
i m not a fan	4468
i m not sure	4215

Table 7.5.: Top 10 most generated responses with respective occurence frequencies when using the last OpenSubtitles snapshot on the test dataset.

Totale Anzahl sätze angeben (können wir ja berechnen?)

make tables the same width!

Sentence	Frequency
i m not sure if i m being sarcastic or not .	17486
i think it s a bit of a stretch .	13058
i m not sure if you re being sarcastic or not .	11647
i m not sure if i m a <unknown> or not .	8307
i m not sure if you re joking or not .	7932
i was thinking the same thing .	7579
<unknown>	6210
i m not sure if i m going to watch this or not .	4257
i m not sure if i m a fan of the show , but i m pretty sure that s a <unknown> .	3232
i m not sure if i m going to watch it or not .	3079

Table 7.6.: Top 10 most generated sentences with respective occurence frequencies when using the last Reddit snapshot on the test dataset.

Totale Anzahl sätze angeben (können wir ja berechnen?)

make

As seen in the both tables above, there are certain responses which are generated a lot of time and are pretty generic and meaningless. Because of that, we thought it would be a good idea to evaluate the models under the Sent2Vec metric one more time, but this time with the top n generic sentences filtered out. We did this with the hope that the generic sentences are the cause of the small average similarity. The results of the analysis with the top n sentences filtered out can be found in Table 7.7 and 7.8. For this analysis, we have only used the Wikipedia model 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 7.7.: 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 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 7.8.: 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 in the tables above, the filtering of the most used responses does not help a lot when it comes to the Sent2Vec evaluation.

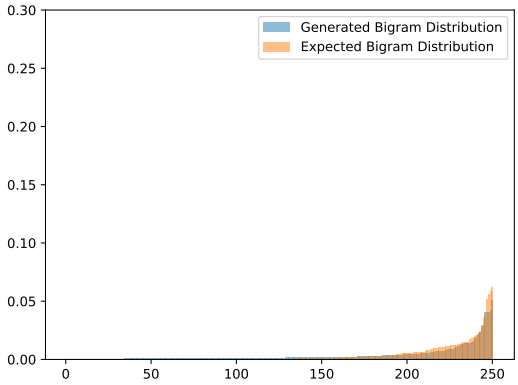
Mixed Feelings about Performance Metrics As seen in this chapter, the performance metrics used to evaluate the models tell us a mixed story about the resulting models. On

one hand, we see that the training went fine and the learning process run as expected. However, when we then test these trained models against the test datasets with the different metrics, it looks like the performance got worse and worse over the time of the training. This is not true in our subjective opinion after "talking" to both models for a prolonged period of time. We think the biggest problem for the evaluation are the generic responses both models seem to generate much more than actual answers. To find the cause of this generic responses, we will now try to analyze the language model which both models have learned while training and try to establish a connection between the language model in the datasets and the ones produced by the models.

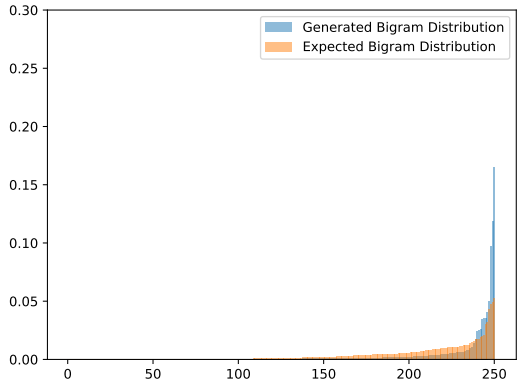
7.3. Language Model & Semantic Understanding

As said at the end of the previous chapter, in this chapter we are going to investigate into the language models the trained models produce and try to get a grasp on how much semantic understanding the models have. For the first analysis, we are going to create compare n-gram distributions over the datasets and compare them with the distributions found in the responses when evaluating the models. We also take a look at the most use n-grams to find a reason why the models produce so much generic sentences. After that, we are going to investigate into how the models actually process the input sequences by first taking a look at the generated thought vectors. We will then go on and try to evaluate if the attention mechanism (see Chapter 3.3) really helps the models when producing responses.

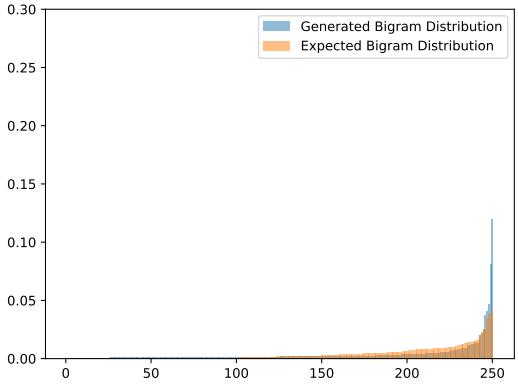
Uni- and Bigram Distributions over Time As the first step, we are going to compare the uni- and bigram distributions of the training datasets with the distributions produced when evaluating the models with the test datasets. For this purpose, we generated unigram and bigram statistics using `nltk` over the training datasets and outputs generated by the models. First, let us analyze the bigram distributions which can be seen in the Figures 7.7 and 7.8. As seen there, at the beginning of the training (i.e. snapshots 0.5M), the bigrams are distributed pretty evenly. However, as longer as the training continues the more right-leaning the distributions of the bigrams in the outputs of the models get. This seems to coincide with the results found in the previous chapter, that the models start to use less and less bigrams but simultaneously increases the usage frequency of often used bigrams. It is also understandable from a stand point that the models learn, which are the important bigrams and which are not. This is especially apparent in the case of the OpenSubtitles model, where we also witness a big discrepancy between the expected and the generated distributions. However, the development of the distribution in the case of the Reddit model looks quite different. It also becomes more right-leaning as the training advances, but it much better fits the expected distribution, for example bigrams in the snapshot 1.5M have almost the same distribution as in the training data.



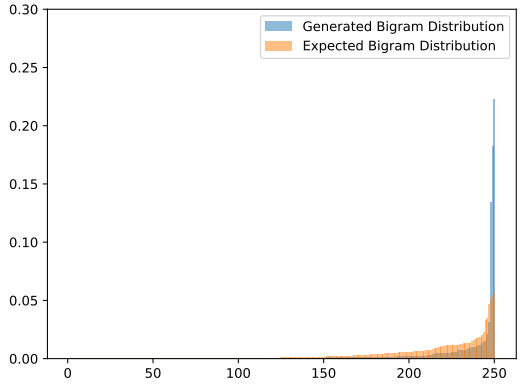
Snapshot 0.5M



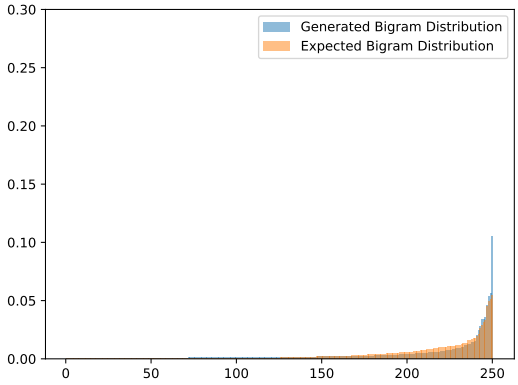
Snapshot 1.0M



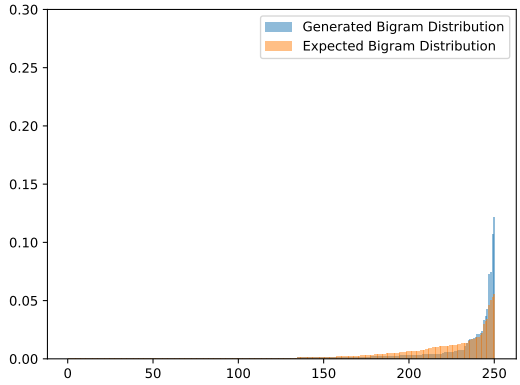
Snapshot 1.5M



Snapshot 2.0M

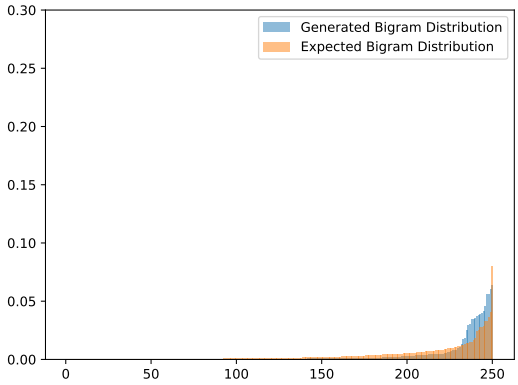


Snapshot 2.5M

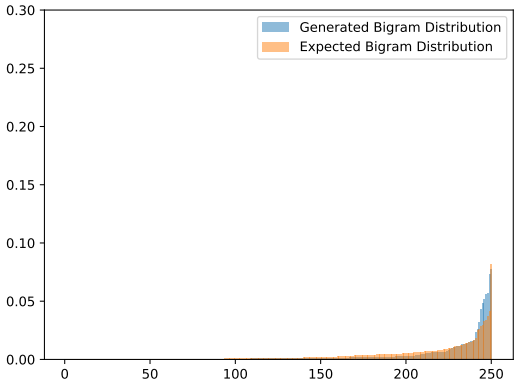


Snapshot 3.0M

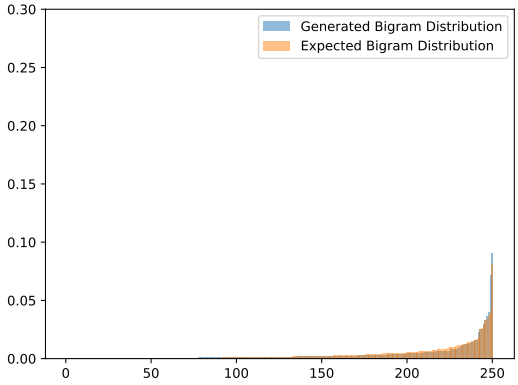
Figure 7.7.: Comparison of the distributions of the top 100 most used bigrams for the responses of the OpenSubtitles models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.



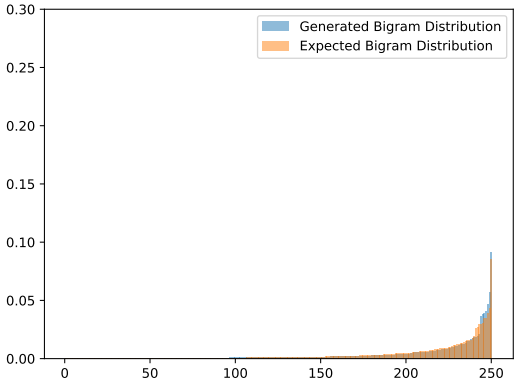
Snapshot 0.5M



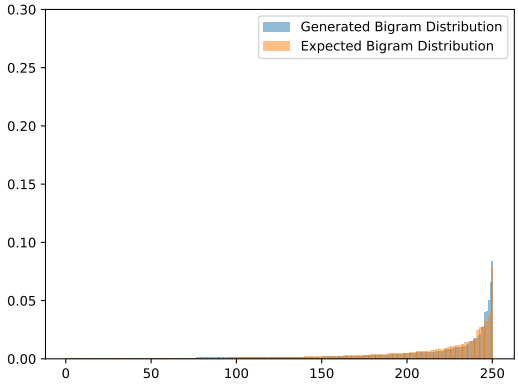
Snapshot 1.0M



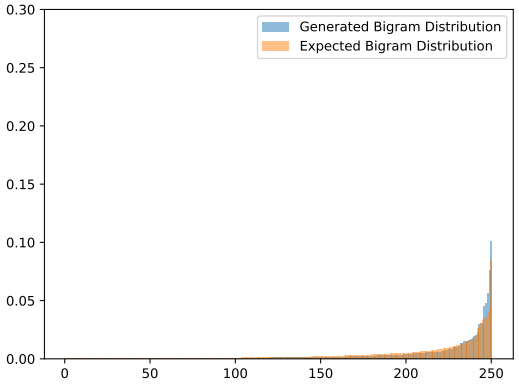
Snapshot 1.5M



Snapshot 2.0M

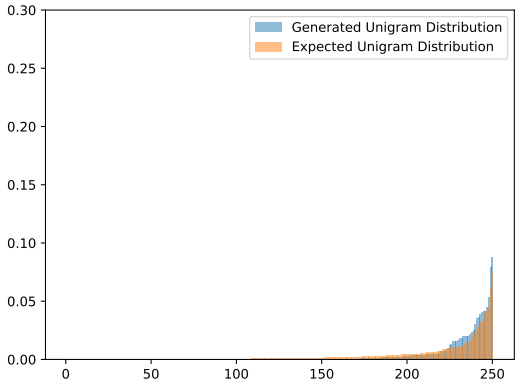


Snapshot 2.5M

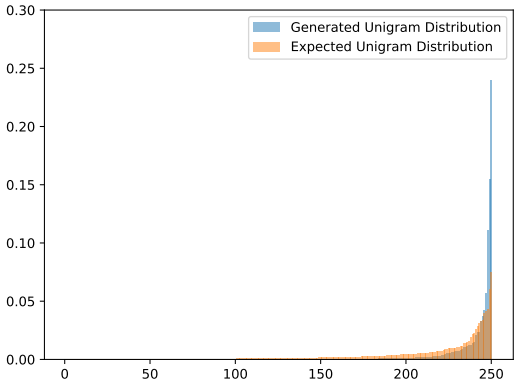


Snapshot 3.0M

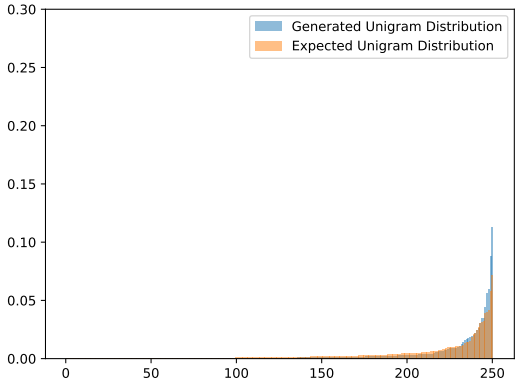
Figure 7.8.: Comparison of the distributions of the top 100 most used bigrams for the responses of the Reddit models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.



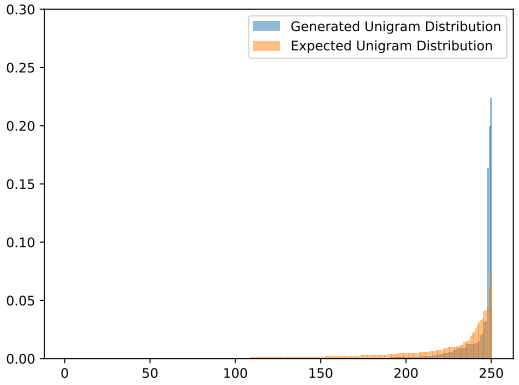
Snapshot 0.5M



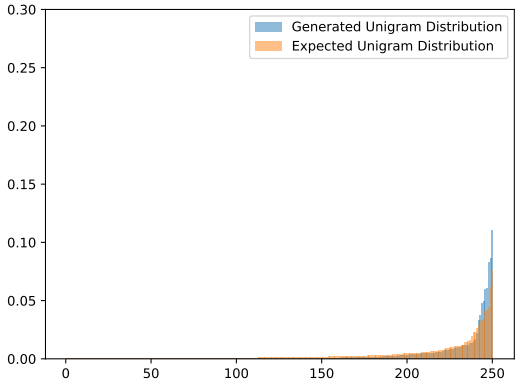
Snapshot 1.0M



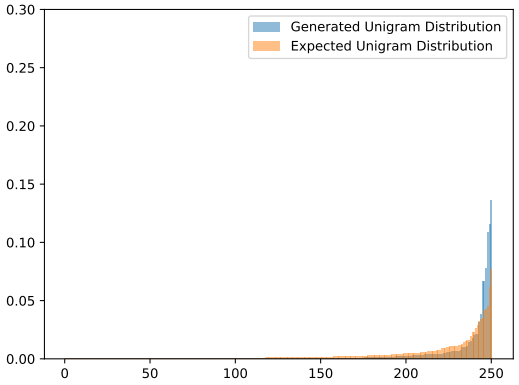
Snapshot 1.5M



Snapshot 2.0M

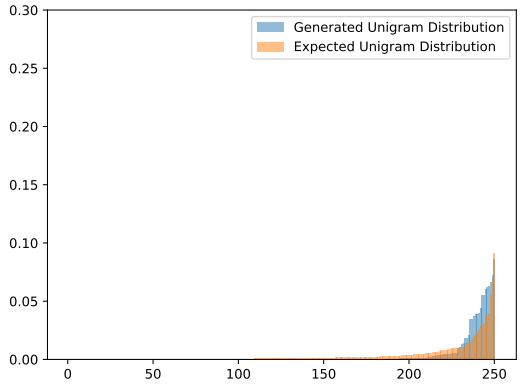


Snapshot 2.5M

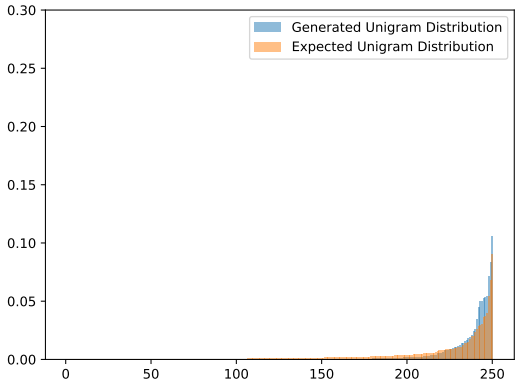


Snapshot 3.0M

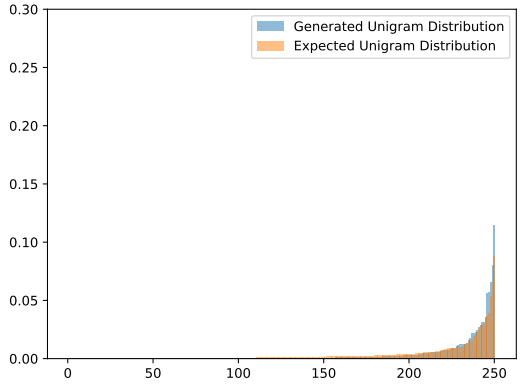
Figure 7.9.: Comparison of the distributions of the top 100 most used unigrams for the responses of the OpenSubtitles models (orange) when using the test dataset and the distribution within the training data (blue). The distributions are compared for each snapshot available.



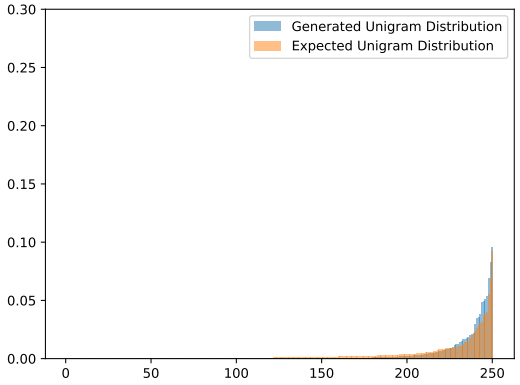
Snapshot 0.5M



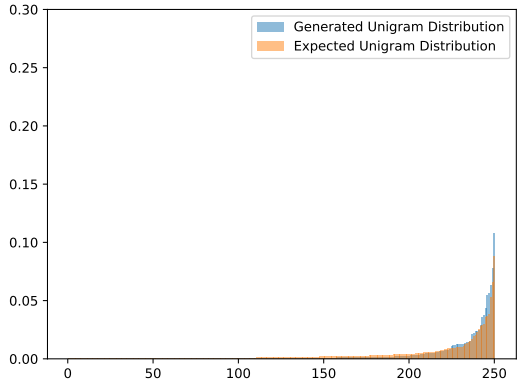
Snapshot 1.0M



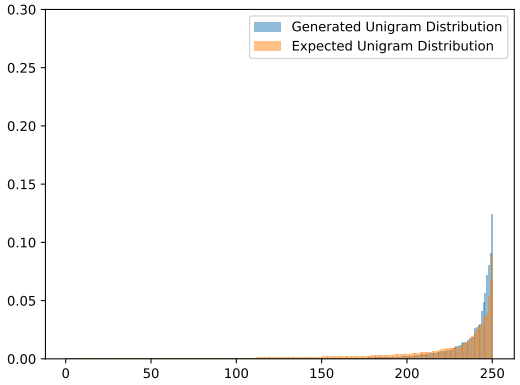
Snapshot 1.5M



Snapshot 2.0M



Snapshot 2.5M



Snapshot 3.0M

Figure 7.10.: 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.

7.3.1. Bi-Gramm

In den Grafiken ?? und ?? wird dargestellt (erklären, was genau man sieht, muss genau sein, da nicht ganz einfach.)

Die bi-gramme und deren Wahrscheinlichkeiten werden übernommen.

grafiken interpretiere

The distribution is right-handed of the words... Für beide Modelle gilt: Starten gleichverteilter, als expected, nähern sich dann dem expected Werten an. Dieser Virgang scheint von links nacht rechts stattzufinden. Das scheint insofern nachvollziehbar, dass er schnell merkt, welche bi-gramme nicht oft vorkommen und entsprechend diese fast nie verwendet.Jedoch herauszufinden, welches der top n n-gramme das Richtige ist, ist ein deutlich schwierigerer Lernprozess. Wir sehen im groben zumindest eine Annäherung der Verteilung. Wünschenswert wäre natürlich hier eine ähnliche Verteilung wie expected, wobei vor allem die weniger häufigen Bigramme für eine vielfältige Sprache stehen. Dies würde sich ganz generell als mass für die Sprachvielfalt eigenen, indem man z.B. die umgekehrte Wahrscheinlichkeit als Gewicht Einfliessen lässt.

7.3.2. Uni-gramm/Wörter

grafiken erstellen und referenzieren und beschrei

Die Uni-gramm und deren Wahrscheinlichkeiten werden übernommen.

The distribution is

grafiken interpretiere

7.3.3. Does the Model have an understanding of natural language sentiment?

grafiken interpretiere

Thought Vectors for Input Sequences After the encoder has processed the whole input sequences, it pass it forward to the decoder to construct the output sequence (see Chapter 3.3). It is the only direct connection the encoder and decoder have in such a model, which means, that the encoder has to "encode" all the information into this thought vector before passing it to the decoder. The thought vector hence represents an embedding of the input sequence in an n dimensional vector space, where n stands for the size of the thought vector. To analyze this embeddings, we collected them for 15 different sample sentences and projected them via PCA into two dimensional space, the results of this projection can be see in Figure 7.11 and 7.12 below.

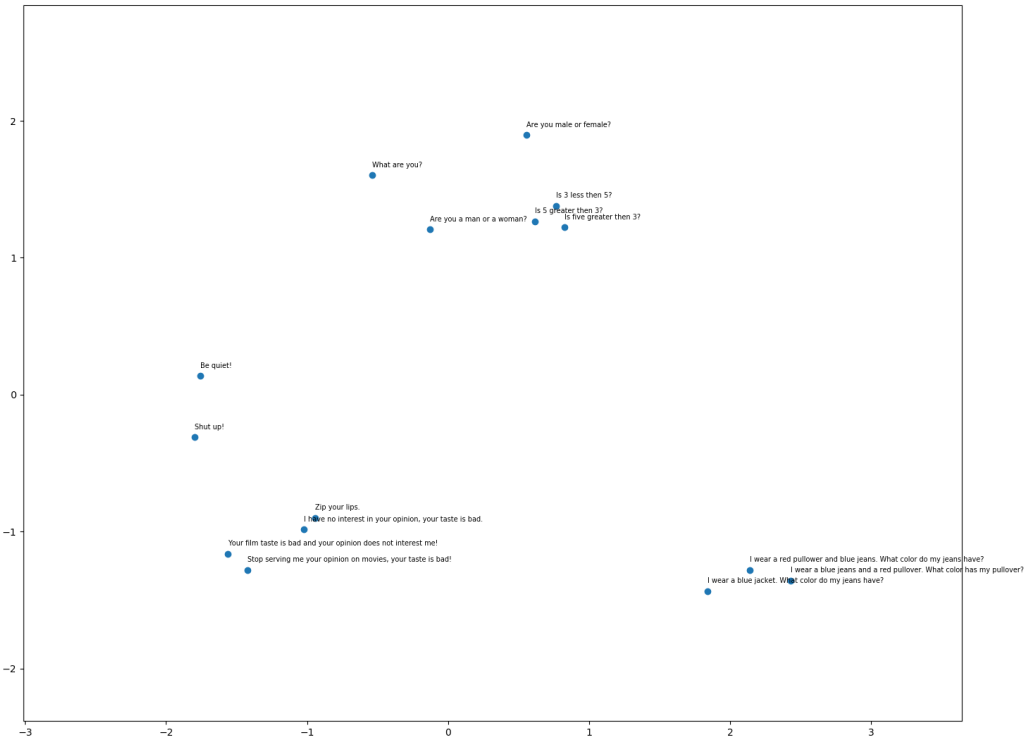


Figure 7.11.: The projected thought vectors for 15 different sentences when using the OpenSubtitles model. PCA was used for the projection.

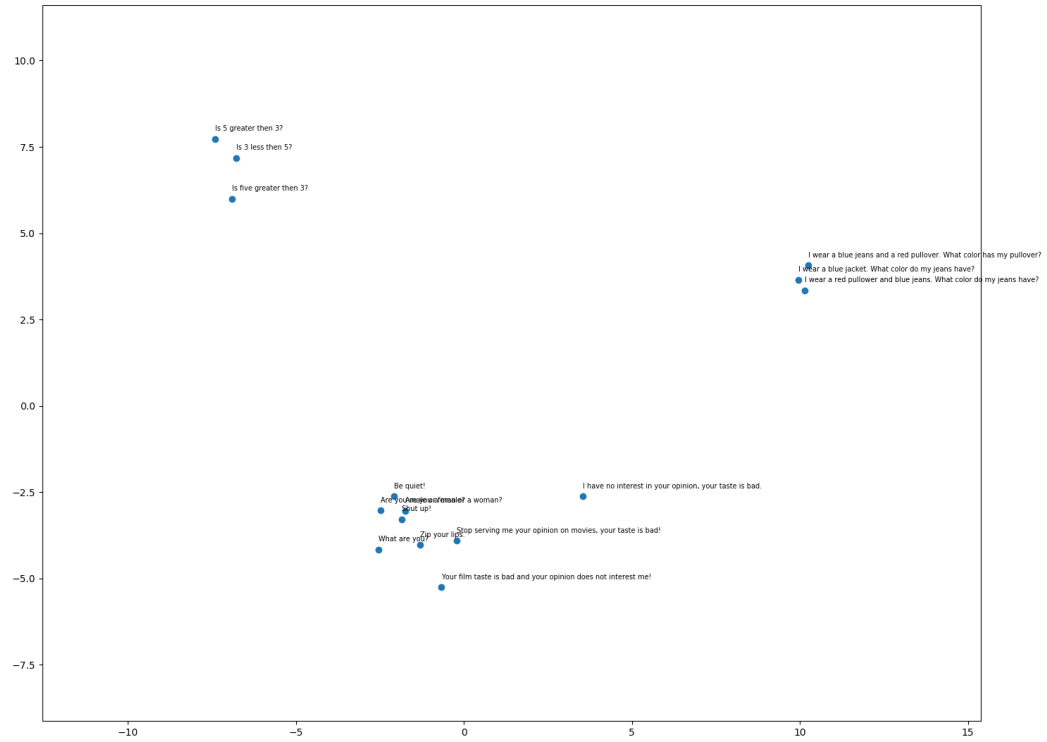


Figure 7.12.: The projected thought vectors for 15 different sentences when using the Reddit model. PCA was used for the projection.

Both of the models seem to have no problems understanding clear, direct sentences where the intent is clear (e.g. "I have no interest in your opinion on movies, your taste is bad!"). This can be seen because similar sentences are clustered together 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. For example, 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 quite close together. But what is interesting to see is that the Reddit model embeds the sentences with curses close to the sentences regarding the gender.

write more

7.4. What Language Model do the Models produce?

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec

ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7.5. Does the Model have an understanding of natural language?

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7.6. How can we fix the detected problems?

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7.7. Generated outputs over time

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor.

Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7.8. Comparison with CleverBot and "Neural Conversational Model"

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

7.9. Reverse Input Feeding

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

8. Conclusion

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

9. Future Work

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

Appendix

A. Neural Networks

Neural networks, in the following referred to as NN, 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're going to explain the functions of all the components which make up a NN.

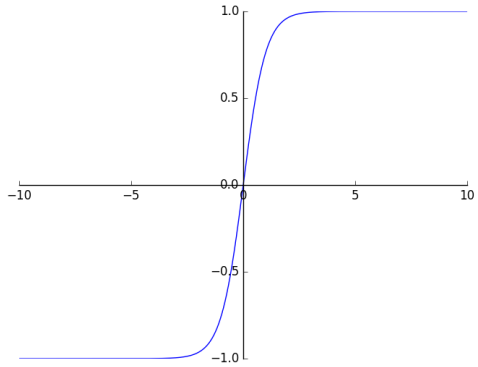
Neuron A NN, at the lowest level, is composed by *neurons*, sometimes also called *perceptrons*. These neurons are basically modelling a mathematical functions and are the building blocks of every NN.

These neurons accept n input values $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ and use them to compute a single output value o. A unique weight w_n from the set $\mathbf{w} = \{w_0, w_1, \ldots, w_n\}$ 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 the called the *bias* value and allows the modelled function to affine instead of linear. This increases the modelling power of such neurons, as the space of functions which can possibly be modelled 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:

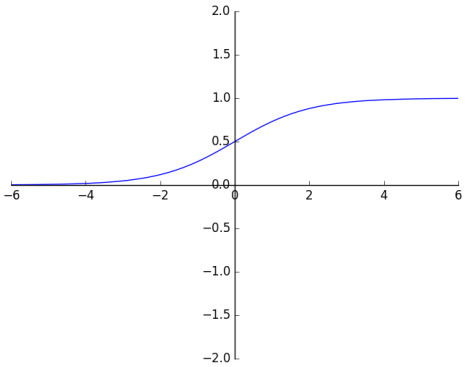
$$o = \varphi(\mathbf{w} \cdot \mathbf{x}) = \varphi\left(\sum_{i=0}^{n} w_i x_i\right) \tag{A.1}$$

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 **??**.

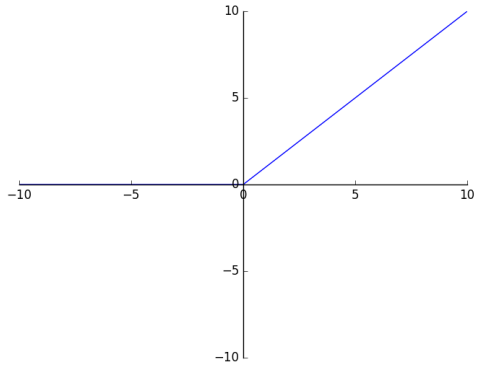
Layer With the neurons introduced one paragraph earlier, we can now start to build the layers of a NN. Each layer consists of multiple neurons stacked on top of each other, as seen in figure A.2. These layers are then arranged in a sequential manner to form a full NN. Each NN usually has at least three of these layers: The first one is called the *input layer*, the second the *hidden layer* and the most right one is the *output layer*.



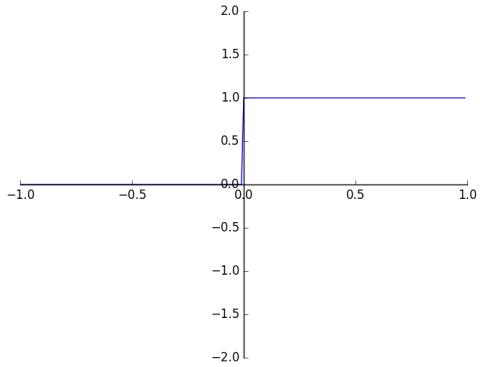
(a) $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



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



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



(d) $\text{binarystep}(x) = \left\{ \begin{array}{ll} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{array} \right\}$

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

The input data $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is 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 $o_{21}, o_{22}, \ldots, o_{2n}$. 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 $o_{31}, o_{32}, \ldots, o_{3m}$ 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*. 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

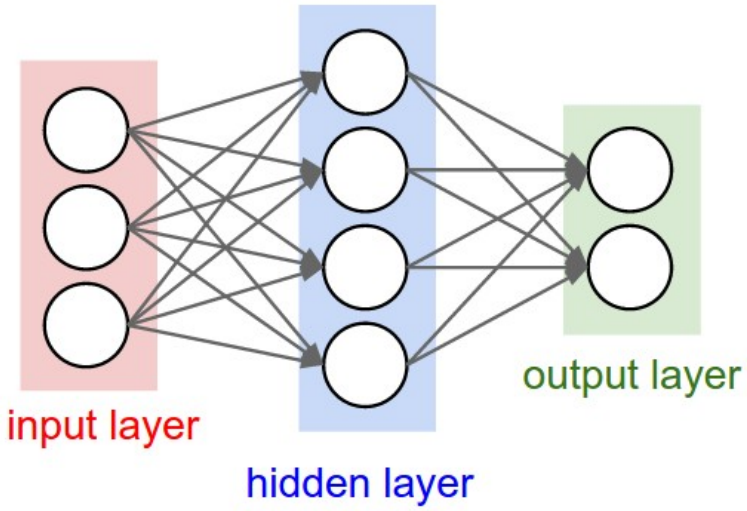


Figure A.2.: Simplified visualization of a NN with one input, hidden and output layer.

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* $\text{msd}(y_{true}, y_{pred}) = \frac{1}{n}\sum_{i=0}^{n}(y_{pred} - y_{true})^2$ or the one used for the model of this thesis, the *categorical cross-entropy* function $H(p,q) = -\sum xp(x)\log(q(x))$. 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 $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ as described above.

2. Use the predicted and expected values in combination with the defined loss function to quanitfy 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:

$$w_{li} \coloneqq w_{li} - \eta \frac{\delta E(\mathbf{w})}{\delta w_{li}} \tag{A.2}$$

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 $\mathbf{w}$:

$$\mathbf{w} \coloneqq \mathbf{w} - \eta(\nabla_{\mathbf{w}} E(\mathbf{w})) \tag{A.3}$$

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 optimium 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* [10] or *AdaDelta* [38], which adapt the learning rate according to the updates done in the past. A comprehensible overview of different gradient-descent algorithms and their variations, including the two mentioned before, can be found in [28].

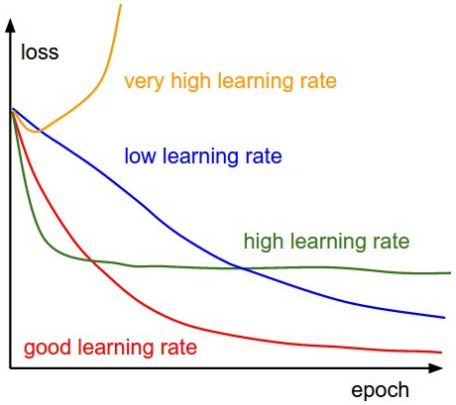


Figure A.3.: Visualization of the development of a loss function when using different learning rates.[1]

[1] http://cs231n.github.io/assets/nn3/learningrates.jpeg

B. Additional Charts

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Etiam lobortis facilisis sem. Nullam nec mi et neque pharetra sollicitudin. Praesent imperdiet mi nec ante. Donec ullamcorper, felis non sodales commodo, lectus velit ultrices augue, a dignissim nibh lectus placerat pede. Vivamus nunc nunc, molestie ut, ultricies vel, semper in, velit. Ut porttitor. Praesent in sapien. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Duis fringilla tristique neque. Sed interdum libero ut metus. Pellentesque placerat. Nam rutrum augue a leo. Morbi sed elit sit amet ante lobortis sollicitudin. Praesent blandit blandit mauris. Praesent lectus tellus, aliquet aliquam, luctus a, egestas a, turpis. Mauris lacinia lorem sit amet ipsum. Nunc quis urna dictum turpis accumsan semper.

C. 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.

C.1. Download

The code of the software system can be downloaded by using `git`[1]. The repository is located in the publicly accessible GitHub server of the ZHAW[2].

C.2. Requirements

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

- `python` in the version 3.4.3[3]
- If a GPU is used:
  - Nvidia GPU driver[4] for the respective GPU.
  - Nvidia `cuda` 8 toolkit[5].

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 following `python` libraries must be installed:

- `numpy` Version 1.11.1
- `theano` Version 0.8.2
- `keras` Version 1.1.0
- `nltk` Version 3.2.1

[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

- `scikit_learn` Version 0.18
- `matplotlib` Version 1.5.3
- `gensim` Version 0.12.4
- `h5py` Version 2.6.0
- `flask` Version 0.11.1

update

The libraries can be installed via the `python` package manager `pip`[6]. It is recommended to us 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 find with newer version without any problems.

[6] https://packaging.python.org/installing/

C.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
`configs/`	In this directory, all the JSON configurations for all experiments are stored.
`report/`	All documents related to the thesis itself are stored in this directory.
`results/`	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.
`scripts/`	The scripts used in this thesis are stored in this directory (see Chapter **??**).
`source/`	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 **??**.

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

update

C.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
`analyse_ngram.py_from_corpus.py`	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 !!REFERENZ RESULTATE!!.
`analyze_timestamp_problematic_opensubtitles.py`	With this script, the time-lag analysis between utterances in the OpenSubtitles corpus can be done (see Chapter 5.6).
`analyze_word_coverage.py`	This script allows to analyse the word coverage of a corpus with regard to a given vocabulary (see Chapter 5.4).
`split_corpus.py`	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).
`evaluate_trained_model.py`	This script allows to evaluate a trained model on a certain dataset and stores the resulting metrics of this evaluation.
`generate_s2v_sequence_embeddings.py`	This script allows for generating Sent2Vec embeddings for the given list of sequence samples. These are then used in Chapter !!!RFERENZ RESULTATE!! for the similarity analysis.
`get_internal_embeddings_from_samples.py`	This script allows to generate "thought vectors" for list of given samples. They are used in the analysis in Chapter !!REFERENZ!!.
`preprocess_opensbutitles_data.py`	This script is responsible for preprocessing the raw OpenSubtitles corpus as described in Chapter 5.3.
`preprocess_reddit_corpus.py`	This script is responsible for preprocessing the raw Reddit corpus and building the tree which is the finally converted into the datasets as described in Chapter 5.3.
`talk_to_model.py`	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 C.6).

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

update

There are much more scripts than ones described above (e.g. for plotting, creation of vocabularies). Especially. 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
`analyse_ngram.py_from_corpus.py`	python scripts/analyse_ngram.py_from_corpus.py data/opensubtitles/opensubtitles_raw.txt 2 results/opensubtitles/bigram_analysis.csv results/opensubtitles/bigram_analysis_words.csv
`analyze_timestamp_problematic_opensubtitles.py`	python scripts/analyze_timestamp_problematic_opensubtitles.py data/OpenSubtitles2016 analysis_timestamps_opensubtitles.json
`analyze_word_coverage.py`	python scripts/analyze_word_coverage.py data/reddit/reddit_corpus.txt word_coverage_reddit_new.json data/reddit/vocab_100k.pickle,data/reddit/vocab_50k.pickle
`split_corpus.py`	Tpython 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
`evaluate_trained_model.py`	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
`generate_s2v_sequence_embeddings.py`	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
`get_internal_embeddings_from_samples.py`	python scripts/get_internal_embeddings_from_samples.py samples.txt results/reddit/model-100000.chkp results/reddit/samples_embeddings.h5
`preprocess_opensubtitles_data.py`	python scripts/preprocess_opensubtitles_data.py data/opensubtitles/raw-xml-files/ data/opensubtitles/opensubtitles_raw.txt
`preprocess_reddit_corpus.py`	python script/preprocess_reddit_corpus data/reddit/full_corpus/ 2014,2015 movies
`talk_to_model.py`	python scripts/talk_to_model.py

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

update

C.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
`device`	`/gpu:0`	Defines the device on which the computations are run.
`train`	`true`	Defines whether a training should be run or not. Is set to `false` in case inference is run.
`git_rev`	`null`	This parameter stores the SHA1 of the latest `git` revision when an experiment was started.
`model_path`	`null`	This parameter signifies if an already trained model should be loaded before starting the training or inference.
`training_data`	`null`	Defines which dataset should be used for training the model while training.
`validation_data`	`null`	Defines which dataset should be used for validating the model while training.
`reverse_input`	`false`	Defines whether the input sequence should be fed to the model in reverse if set to `true`.
`use_last_output_as_input`	`false`	Defines whether the output of the last sample should be used as the input for the next sample as described in Chapter 6.
`start_training_from_beginning`	`false`	Defines whether the training should start from the first sample of the training dataset or not. If it is set to `true`, 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.
`show_predictions_while_training`	`false`	Defines whether the outputs of the model should be printed to the terminal when training.
`show_predictions_while_training_num`	`5`	Defines how much predictions should be printed at the end of each epoch when training.
`vocabulary`	`null`	Defines which `pickle` vocabulary should be used for the current model.
`epoch`	`1`	Defines the number of epochs which should be done in case of training a model.
`save_model_after_n_epochs`	`10`	Defines the interval in which the model is stored based on the number of epochs finished.
`epochs_per_validation`	`10`	Defines the interval in which the model is validated while training based on the number of epochs finished.
`batches_per_validation`	`0`	Defines the interval in which the model is validated while training based on the number of epochs finished.
`batches_per_epoch`	`1000`	Defines how much batches should be processed in one epoch.
`batch_size`	`1`	Defines how much samples should be put in one batch.

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

Name	Default Value	Description
max_input_length	50	Defines the maximum number of words to consider from the input sequence.
max_output_length	50	Defines the maximum number of words the decoder can generate before the decoding stops.
num_encoder_layers	1	Defines the number of layers used for the encoder.
num_decoder_layers	1	Defines the number of layers used for the decoder.
num_hidden_units	1024	Defines the size of the hidden state used in the encoder and decoder cell.
cell_type	LSTM	Defines which kind of RNN cell is used for the model. Can be either LSTM, GRU or RNN.
sampled_softmax_number_of_samples	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.
hidden_state_reduction_size	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.
max_random_embeddings_size	512	Defines the dimensionality of the word embeddings used within the model.
use_beam_search	false	Defines whether the beam search decoder should be used instead of the greedy decoder.
beam_size	10	The number of beams which should be considered when using the beam search decoder.
beam_search_only_best	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.
buckets	[[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 max_input_length and max_output_length.
dropout_input_keep_prob	1.0	Defines how much percent of the input to the RNN cells should be kept when using dropout.
dropout_input_keep_prob	1.0	Defines how much percent of the output of the RNN cells should be kept when using dropout.

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

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:

```
{
  "epochs": 150000,
  "batches_per_epoch": 100,
  "batches_per_validation": 2500,
  "epochs_per_validation": 50,
  "save_model_after_n_epochs": 100,
  "batch_size": 64,
  "cell_type": "LSTM",
  "num_hidden_units": 2048,
  "hidden_state_reduction_size": 1024,
  "num_encoder_layers": 1,
  "num_decoder_layers": 0,
  "training_data": "data/reddit_train.txt",
  "validation_data": "data/reddit_valid.txt",
  "vocabulary": "data/reddit/vocab_50k.pickle",
  "max_random_embeddings_size": 1024,
  "max_input_length": 30,
  "max_output_length": 30,
  "reverse_input": false,
  "show_predictions_while_training": true,
  "buckets": [[30, 30]],
  "sampled_softmax_number_of_samples": 512,
  "word_tokenizer": "none",
  "use_last_output_as_input": true,
  "start_training_from_beginning": false
}
```

Figure C.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`.

C.6. Web Frontend

We implemented a simple web frontend for communicating with already trained models. It is implemented by using `flask`, `jQuery` and the `bootstrap` frontend framework. To us the frontende, one has to start it using the following command:

```
$ 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 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 `true` there. For a comprehensive list of all configuration values and their meaning, please see Chapter C.5 or the source code itself.

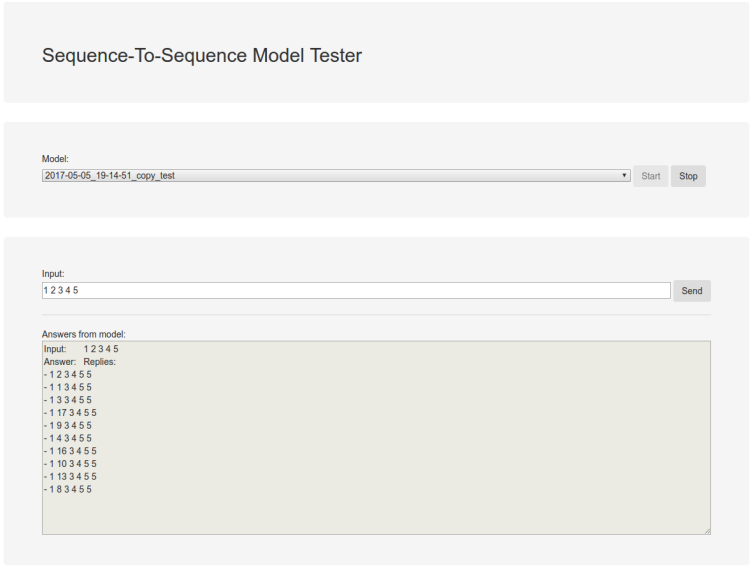


Figure C.2.: Frontend showing the output when sending the sequence “1 2 3 4 5” to a model trained on the copy task (see chapter 4.3).

List of Tables

List of Figures

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, *TensorFlow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. [Online]. Available: `http://tensorflow.org/`.

[2] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *ArXiv preprint arXiv:1409.0473*, 2014.

[3] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[4] S. Bird, E. Klein, and E. Loper, *Natural language processing with python: Analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[5] 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.

[6] 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.

[7] 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.

[8] R. Collobert, K. Kavukcuoglu, and C. Farabet, "Torch7: A matlab-like environment for machine learning," in *BigLearn, NIPS Workshop*, 2011.

[9] R. Desimone and J. Duncan, "Neural mechanisms of selective visual attention," *Annual review of neuroscience*, vol. 18, no. 1, pp. 193–222, 1995.

[10] 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.

[11] A. Graves, "Generating sequences with recurrent neural networks," *ArXiv preprint arXiv:1308.0850*, 2013.

[12] 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.

[13] 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.

[14] D. von Grünigen, M. Weilenmann, J. Deriu, and M. Cieliebak, "Potential and limitations of cross-domain sentiment classification," *SocialNLP 2017*, pp. 17–25, 2017.

[15] 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)*, Pasadena, CA USA, Aug. 2008, pp. 11–15.

[16] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen," Diploma Thesis, Institut für Informatik, Lehrstuhl Prof. Brauer, Technische Universität München, 1991.

[17] S. Hochreiter and J. Schmidhuber, "Ltsm can solve hard time lag problems," in *Advances in Neural Information Processing Systems: Proceedings of the 1996 Conference*, 1997, pp. 473–479.

[18] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[19] 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.

[20] 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`.

[21] N. Kalchbrenner and P. Blunsom, "Recurrent continuous translation models.," in *EMNLP*, vol. 3, 2013, pp. 413–422.

[22] 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.

[23] V. Mnih, N. Heess, A. Graves, *et al.*, "Recurrent models of visual attention," in *Advances in neural information processing systems*, 2014, pp. 2204–2212.

[24] H. Nguyen, D. Morales, and T. Chin, "A neural chatbot with personality," [Online]. Available: `http://web.stanford.edu/class/cs224n/reports/2761115.pdf`.

[25] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised Learning of Sentence Embeddings using Compositional n-Gram Features," *ArXiv*, 2017. arXiv: `1703.02507 [cs.CL]`.

[26] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks.," *ICML (3)*, vol. 28, pp. 1310–1318, 2013.

[27] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora," English, in *PROCEEDINGS OF THE LREC 2010 WORKSHOP ON NEW CHALLENGES FOR NLP FRAMEWORKS*, `http://is.muni.cz/publication/884893/en`, Valletta, Malta: ELRA, May 2010, pp. 45–50.

[28] S. Ruder, "An overview of gradient descent optimization algorithms," *ArXiv preprint arXiv:1609.04747*, 2016.

[29] 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.

[30] 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.

[31] Theano Development Team, "Theano: A Python framework for fast computation of mathematical expressions," *ArXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: `http://arxiv.org/abs/1605.02688`.

[32] O. Vinyals, Ł. 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.

[33] O. Vinyals and Q. Le, "A neural conversational model," *ArXiv preprint arXiv:1506.05869*, 2015.

[34] S. v. d. Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: A structure for efficient numerical computation," *Computing in Science & Engineering*, vol. 13, no. 2, pp. 22–30, 2011.

[35] P. J. Werbos, "Backpropagation through time: What it does and how to do it," 10, vol. 78, IEEE, 1990, pp. 1550–1560.

[36] 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.

[37] 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.

[38] M. D. Zeiler, "Adadelta: An adaptive learning rate method," *ArXiv preprint arXiv:1212.5701*, 2012.