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

19. Mai 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, ~~in~~ wie sich ein einzelner Sentiment-Klassifizierer auf verschiedenen Domänen verhält und ob es Sinn macht, Datensätze mehrerer Domänen in Kombination zu verwenden.

Noch Kommasetzung und Gross/ Kleinschreibung korrigieren (lassen)

Vorschlag: "Wir verwenden Texte aus den Domänen "Reddit-Threads" und "Film-Untertitel", um zu analysieren, wie stark der Einfluss unterschiedlicher Word-Embeddings und Distant-Phasen auf einen Sentiment-Klassifizierer ist und ob es vielleicht eine Kombination aus Word-Embeddings und Distant-Phasen gibt, die für beide Domänen optimal ist. Die Resultate zeigen, dass es so eine optimale Kombination aus Embedding und Distant-Phase nicht gibt." Ich weiss, dass ihr gelernt habt, dass man das Passiv verwenden soll, aber das gilt nicht für die Informatik, die in ihrem Schreibstil stark angelsächsisch geprägt ist. Dort herrscht das Aktiv vor.

Hier ein Ausschnitt aus George Orwell's "Politics and the English Language":

A scrupulous writer, in every sentence that he writes, will ask himself at least four questions, thus:

1. What am I trying to say?
2. What words will express it?
3. What image or idiom will make it clearer?
4. Is this image fresh enough to have an effect?

And he will probably ask himself two more:

1. Could I put it more shortly?
2. Have I said anything that is avoidably ugly?

Und noch einer:

I think the following rules will cover most cases:

1. Never use a metaphor, simile, or other figure of speech which you are used to seeing in print.
2. Never use a long word where a short one will do.
3. If it is possible to cut a word out, always cut it out.
4. Never use the passive where you can use the active.
5. Never use a foreign phrase, a scientific word, or a jargon word if you can think of an everyday English equivalent.
6. Break any of these rules sooner than say anything outright barbarous.

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

Im Englischen wird weniger mit Bindestrichen gearbeitet, als im Deutschen. Also "sentiment classification" und "distant phase"

In the following work we ~~are going to~~ investigate the potential for crossdomain sentiment-

Vorschlag: "The main research question is whether it is possible to use data form one domain to train or improve a sentiment classifier that will be used on another domain. If it turns out to be possible, we could use carefully curated data from one domain to improve sentiment classifiers from domains for which training data is hard to obtain or nonexistent."

In the first part, we analyse how well the sentiment-classifiers for different together with several combinations of distant-phases and word-embedding that there's no a priori answer on which combination is the best upfront evaluated on each domain separately.

"multiple" würde hier bedeuten, dass ihr mehrere Kombinationen gleichzeitig verwendet. Aber ihr nehmt ja erst eine Kombination, dann eine andere etc.

Relativsätze mit "that" ohne Komma, also "It is shown that", oder noch besser "We show that"

The next part focuses on how well specialized sentiment-classifiers work on different do-

"For this purpose, we first train several sentiment-classifiers on one domain, and then evaluate them on all others. The results show that there is no single best domain on which to train a generalized classifier, and that the classifier's performance deteriorates when 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." Welchen?

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.

Finally, 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

"The results show that the performance of a generalised sentiment classifier trained on ablation datasets is only slightly worse than the performance of specialized classifers. This makes ablation datasets useful in certain use cases." Welche use cases habt ihr da im Sinn?

Inhaltsverzeichnis

Preface

Danke Stephan, Mark und Jan! :D

1. Introduction

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

2. Related Work

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

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

"We show the basic principles behind RNNs, explain some of the problems they have in practice, and how to modify the them in order to avoid these problems."

models, called *Sequence-To-Sequence* (seq2seq) models [23], 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. ~~Basically~~, 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) as shown by !REFERENZ FÜR UNSUPERVISED LEARNING?!. A basic introduction into the principles of neural networks can be found in Appendix A.

Füllwörter wie "basically", "essentially" etc, kann man IMMER streichen. Gilt auch für "the fact that".

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 ~~stretch~~ by one participant without the second one interjecting.

unbroken sequence

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

Technisch gesehen braucht es für einen Dialog nicht genau zwei Teilnehmer. Die Vorsilbe "Dia-" steht nicht für "zwei" (Latein), sondern laut Wikipedia für "durch" (Griechisch): The term dialogue stems from the Greek διάλογος (dialogos, conversation); its roots are διά (dia: through) and λόγος (logos: speech, reason).

of possibly multiple
ow the term dialog
ontext into account.
capable of handling

Ihr solltet mal alle eure "multiple"s überprüfen und ggfs durch "several"s erzetzen.

context betwee evertheless use he term dialog to describe a string of multiple sam .

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

"Appendix", "Chapter", "Section" etc, gross schreibem. In LaTeX ausserdem mit Tilde ein non-breaking space setzen: "Appendix~\ref{...}"

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 into 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).

Im Englischen nach Doppelpunkt klein schreiben

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 **??**) ~~try to~~ solve this problem by introducing the network, which allows it to exploit information not only from the also from inputs of the past. It does this by transferring a state, usually *state*, through the recurrence between different time steps. In a more formal way, one could say that this recurrence allows RNNs to "exhibit dynamic behaviour across the temporal dimension of the input data".

Is this really "more formal"? It uses more jargon, certainly, but is it really "more formal"?

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

$f\colon R^n \times R^m \rightarrow R^n \times R^m$

the cell.

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

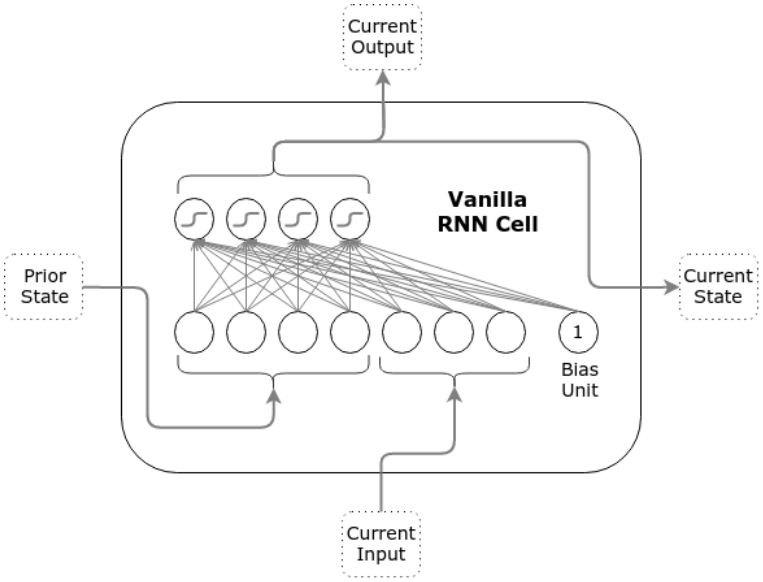


Abbildung 3.1.: Internal structure of a vanilla RNN.[1]

The internal structure is similar to the one of NNs, as explained in appendix A, except for the recurrence which we're going to elaborate on below.

Verkürzungen ("we're", "we'll" etc) nur selten verwenden. Der Normalfall ist das Ausschreiben.

$$\begin{aligned} h_t &= \varphi_h(\mathbf{w}_h \cdot \mathbf{x}_t + \mathbf{u}_h \cdot \mathbf{h}_{t-1} + \mathbf{b}_h) \\ &= \varphi_h\left(\sum_{i=0}^{n} w_i x_{ti} + \sum_{i=0}^{n} u_i h_{t-1i}\right) \end{aligned} \tag{3.1}$$

Second part of equations not correct

Sollte das hier o_t statt y_t sein?

$$\begin{aligned} y_t &= \varphi_y(\mathbf{w}_y \cdot \mathbf{h}_t + \mathbf{b}_y) \\ &= \varphi_y\left(\sum_{i=0}^{n} w_i x_{ti} + \sum_{i=0}^{n} u_i h_{t-1i}\right) \end{aligned} \tag{3.2}$$

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.
- o_t kommt in den Gleichungen icht vor!
- fett? h_t stands for the hidden state at time step t.
- $\mathbf{w}_h$, u_h? $\mathbf{u}$ are the weight matrices learned while training the model.
- $\mathbf{b}_h$ and $\mathbf{b}_o$ 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 o_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 o_{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 o_{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.

Worauf bezieht sich "this"?

To come back to our weather prediction problem, this allows the cell to remember what weather it has seen in the past ~~(e.g. rainy, sunny, high pressure, low pressure)~~ and adjust its future predictions accordingly. The recurrence also solves the problem of time windows of differente 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 5 days or past 5 years, it can compress all this into the hidden state h_t.

"five"

While this is the solution for our problem of different window sizes, it is also a main bottleneck of the model

Hier in LaTeX nicht " nehmen, sondern `` für linke und " für rechte Anführungszeichen. Aber wieso überhaupt Anführungszeichen?

window is past information is 5 years and we process th we would need to remember the informations from arou ve 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" informa en state easily, as ~~the~~ Equation 3.1 only adds to the h urse, 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.

"Information" in Englisch immer im Singular!

Vanishing / Exploding Gradient Problem The recurrent nature of RNNs can cause problems in practice, one of the most problematic is the vanishing/exploding gradients problem: While training such models, we condition the weight matrices $\mathbf{w}_h$, $\mathbf{w}_o$ and $\mathbf{u}$ via backpropagation 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 deriva . The entirety of all these derivatives w.r.t. to the parameters is called the gradient. Since the gradient also flows through the activation functions φ_h of the recurrence, this computations also include the derivatives of this function. Most of the commonly used activation functions today have codomains in $\mathbb{R}$ and take on values in ~~the range of~~ $[-1,+1]$ (e.g. tanh). This means that multiplying many of these derivatives 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 that we

Nicht \frac benutzen, sondern /: $dz/dx = (dz/dy)(dy/dx)$

Typisch deutsch: "which" verwenden, wo "that" gemeint ist. "Which" leitet eine genauere Erklärung des Terms ein, der links davon steht: "This is a RNN, which is a subtype of neural network". "That" führt den Gedanken weiter: "This is a RNN that we use to solve our problem". NB: Komma bei which, aber keins bei that.

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 (e.g., $t > 20$).

In der Kürze liegt die Würze.

The problem of vanishing gradients was first analyzed by Sepp Hochreiter in his diploma thesis [12]. A more formal explanation of the problem, with links to dynamic systems and formal proofs, <-- Wenn man hier ein Komma macht, muss man auch nach "Problem" eins machen lems occur, can be found in [20]. The details on how ent is adapted to RNNs and time-dependent problems *Through Time* or *TBTT* in short, can be found in [26].

Long Short-Term Memory Networks To solve the problem with vanishing/exploding gradients mentioned before, J Nennung der Vornamen nicht üblich introduced an advanced version of a RNN cell, called *Long Short-Term Memory*, or LSTM in short [13]. 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 to forget stored information.
of forgetting

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 Ab hier gibt's endet die Englisch-Lektion :) state h_{t-1}, add to the new hidden state h_t and use for the computa There are three different gates in an LSTM cell (as depicted in figure **??**):

- **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 o_t.

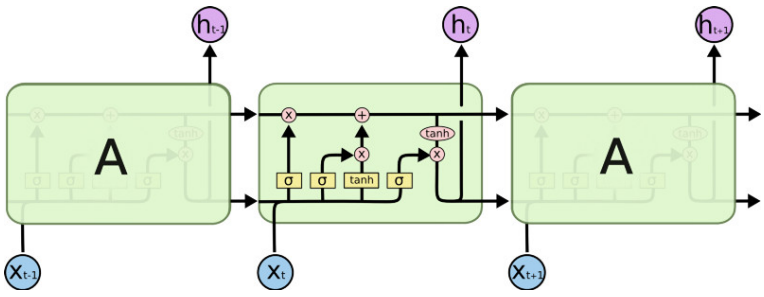


Abbildung 3.2.: Internal structure of a LSTM cell.[2]

[2] 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 [22]. 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. [9]. There are even more architectures for RNN cells like *Gated Recurrent Unit* [5] and *Convolutional LSTM* [27], 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 geht doch :) people around the same time [23][16][4], 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 informations on where and how seq2seq models can be 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 **??**). 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 **??**). The tokens are produced by feeding the final state of the decoder cell to a softmax classification layer which outputs probabilities over the words in the vocabulary. It then usually chooses (when using a greedy decoding approach, see paragrahp "Decoding Approaches") 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 current input x_t.

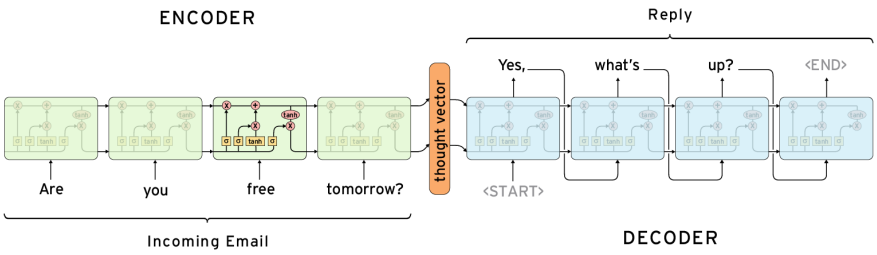


Abbildung 3.3.: Internal structure of a Sequence-To-Sequence Model.[3]

Such a model is also called an *end-to-end* model because it, beeing 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 o do-ne by feeding the network the expected wo ows for gradient-descent to be applied. Formall ing, the model is conditioned to maximize the co t_2), where t_1 is the length of the input sequence and t_2 the length of the corresponding output sequence. As said before, the encoder cell produces a thought vector t, which is nothing more than its hidden state after it ha $x_1, \ldots, x_2$, and passes this to the decoder for generating the output sequence. Hence, the following equation is a formal way of expressing the conditional probability such a model is trained to maximize:

Sind die Symbole t_1 und t_2 Standard? Ich finde das höchst verwirrend. Besser wären m und n oder M und N oder sowas. Denn t_1 und t_2 stehen ja nicht unbedingt in einem kausalen Zusammenhang zueinander

Vermutlich x_1, \dots, x_{t_2}

This equation looks very wrong, if only because the x don't appear on the RHS. Also, aren't you assuming some kind of independence?

$$p(y_1, \ldots, y_{t_1} | x_1, \ldots, x_{t_2}) = \prod_{t=1} p(y_t | t, y_1, \ldots, y_{t-1}) \tag{3.3}$$

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

Is that correct? Check one more time!

If we look at the example depicted in figure **??**, the generation of the output sequence is done as follows:

1. Feed the encoder ~~with~~ 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 $x_1, \ldots, x_{t_1}$ are exhausted ~~when t_1 is the length of the input sequence~~.

3. Pass the hidden state of the encoder cell to the decoder cell after the first has processed the last word of the input sequence x_{t_1}.

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_2 for the output is reached.

The concatenated outputs $y_1, ..., y_{t_2}$ 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 given the context $y_1, \ldots, y_{t-1}$ and the thought vector 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, as done in !!REFERENZEN!!. 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

Ich nicht, ich mach das genau andersrum, damit ich beim Durchlesen schon gleich weiss, worauf ich achten muss: Ich lese erst die Fragen und dann den Text

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.

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 [25] 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:

$$\begin{aligned} u_i^t &= v^T \tanh(W_1 h_i + W_2 d_t) \\ a_i^t &= \operatorname{softmax}(u_i^t) \\ d'_t &= \sum_{i=1} a_i^t h_i \end{aligned} \tag{3.4}$$

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.

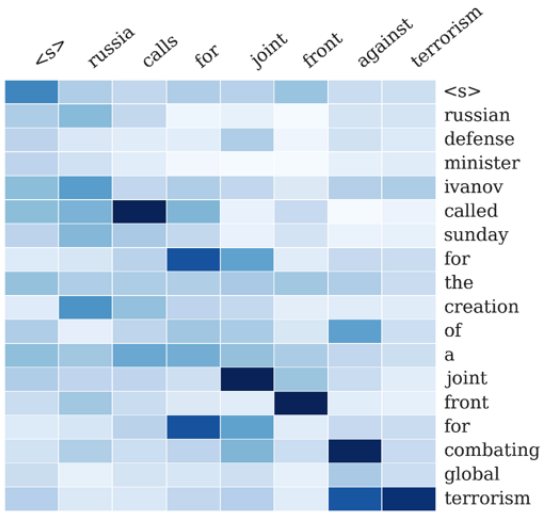
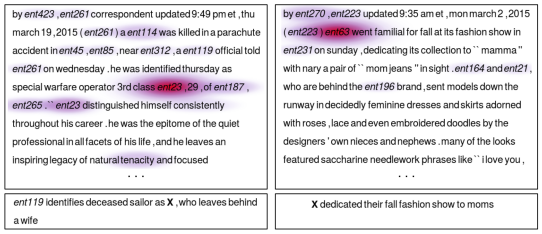


Figure 1: Example output of the attention-based summarization (ABS) system. The heatmap represents a soft alignment between the input (right) and the generated summary (top). The columns represent the distribution over the input after generating each word.

(a) Heatmap visualizing the attention weights.



(b) Attention weights visualized via a weights overlay

Abbildung 3.4.: Examples of how attention weights can be visualized using heatmaps and overlays.

Was heisst "fully" hier? Ich habe das schonmal weiter oben gesehen.

ion 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 selects a random XYZ 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.

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 [7][14][18] and has a wide range of applications there [10][28][3]. 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).

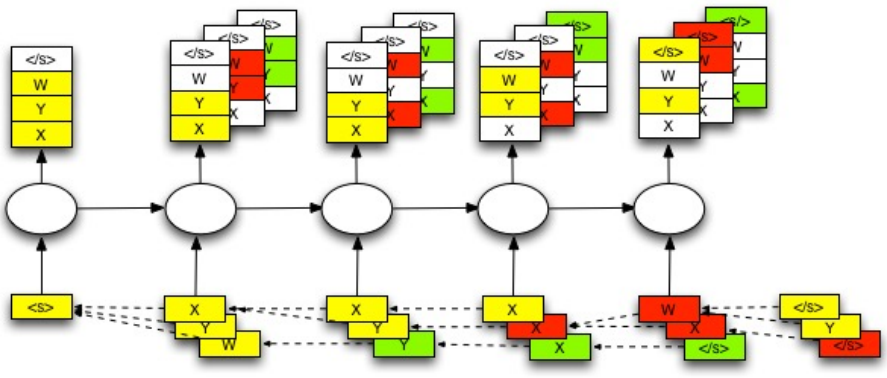


Abbildung 3.5.: Visualization on how beam search works in the context of seq2seq models.[4]

[4]https://github.com/tensorflow/tensorflow/issues/654

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).

$$H(p,q) = -\sum_{X} p(x) log_2(q(x)) \tag{3.5}$$

In the case of sequence-to-sequence learning, the expected probability is ~~effectively~~ 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 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:

$$\text{perplexity}(X) = 2^{H(X)} = 2^{-\sum_X p(x) log_2(q(x))} \tag{3.6}$$

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 \frac im Fliesstext vermeiden, lieber / nehmen: $1/|V|$ abulary size. This does not reflect ~~the~~ 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 umgekehrt: "2 raised to ..." . This means, we effectively have one performance metric to use. To fix this problem, we also investigated

into using Sent2Vec[5] antic difference of
ted by the model when trained. We prop
to test the model:

Gibt es einen Grund für den Schreibmaschinenfont? Habt ihr ja anderswo auch nicht genommen.

Ist ein bisschen blöd, wenn es "similarity" heisst, aber grosse Werte nicht für grosse Ähnlichkeit stehen. Daher "difference"

1. Allocate a variable for storing the sum of distances.
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 euclidean distance between the embeddings in the n-dimensional vector space.
   d) Add this distance to the sum of distances.
3. Divide the sum of distances by the number of samples. This is the final result which can be used as a metric.

This procedure allows us to furthe

Hier kommt dann doch ein Bindestrich rein: n-dimensional

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 small value 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 big to signify that there is a large mismatch between the meaning of the expected and generated sentence.

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

4. Software System

In this chapter, we ~~are going to~~ discuss the development of the software system used to conduct the experiments described later in this thesis. First, we ~~are going to~~ 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 having to put additional efforts into it.

  Das ist nicht "reproducible". "Reproducible" heisst, dass dieselben Ergebnisse rauskommen, wenn man das Experiment wiederholt

- **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**: Du onment in which the experiments were he hardware) is kind of volatile with re design the system in such a way, that the unexpected termination of a running experiment should be easily recoverable. This mea most recent model from the previous trai training was interrupted. This includes lo kipping all of

  Also entweder ganz einfach "unstable" oder mit Understatement "has room for improvement in the area of stability" oder "is not the most stable of platforms" oder sowas :)

  Nein, nicht die "unexpected termination" muss "easily recoverable" sein :-) Was ihr meint, ist eher "we should easily be able to recover from the premature termination of an experiment"

[1] http://www.json.org/

the training data the model was already trained on as well as loading all the metric already collected in the first run.

Aus Merriam-Webster

- Evaluable: The trained models should be easily evaluatable, 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 "speak" to a trained model.

I'd like to speak to a trained model, please :) :)

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 losing any of the results.

to lose out on = etwas verpassen

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.

Schaut mal über euren Text und überdenkt mal, ob ihr nicht "large" statt "big" schreiben wollt.

Switch from Keras to TensorFlow The first and probably also the biggest 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 [11]. 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][24][6].

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

[2] https://keras.io/
[3] https://www.tensorflow.org/

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.

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 (see chapter !!REFERENZ!!) 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 ~~a~~ training in the hope that this would only be ~~only~~ 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

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

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 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 [25], we tried to make our model as big as possible to come as close to the one used by ..., which had 4096 hidden units and 100k vocabulary aused 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 100k 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 [25]), 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.

[6] https://www.tensorflow.org/tutorials/seq2seq

The last part which we implemented for our model was the beam-search. As we did not have ~~not had~~ much ~~prior experience with~~ a computational graph, this task was much harder to pull off th We were lucky and found several implementations for it scattere rnet[78910]. 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.

Dies ist nicht Fussnote Nr 78910! Also Kommas dazwischen :)

Sowas findet man sonst nur beim Kampfsport-Training! Und bei der ZHAW gibt's das ganz umsonst!

Summary In summary, ou 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.

Super Abschnitt, liest sich wie ein Krimi! :D

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.

[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

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 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.

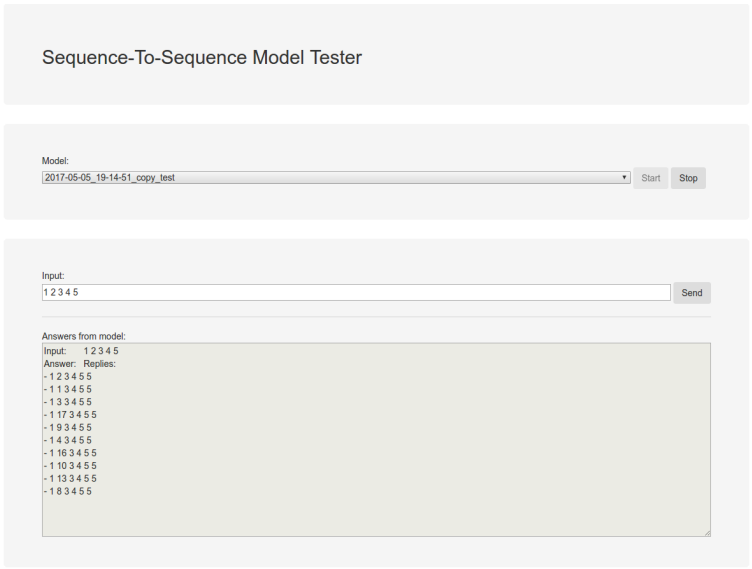


Abbildung 4.1.: Frontend showing the output when sending the sequence "1 2 3 4 5" to a model trained on the copy task (see chapter 4.3).

Da habt ihr irgendwo noch "german" drin

[12]https://github.com/Marsan-Ma/chat_corpus/
[13]http://flask.pocoo.org/
[14]https://jquery.com/
[15]http://getbootstrap.com/

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:

- 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`, version 3.5.2 using

[16] https://www.docker.com/
[17] https://github.com/NVIDIA/nvidia-docker

`TensorFlow`[18] in the version 1.0. For the GPU integration, we have used the Nvidia GPU driver in conjunction with the `cuda`[19] in the version 8.0.

[18] http://tensorflow.org
[19] https://developer.nvidia.com/cuda-toolkit

5. Data

Ich hab hier mal aufgehört, weil mir klar scheint, dass diese Abschnitte noch sehr roh sind. Ich lese gerne weiter, sobald das fortgeschritten ist.

Dieses Kapitel beinhaltet Informationen über die Datensets. Auch erklären wir hier unsere Vorgehensweise, wie wir von den Rohdaten zu unseren "Trainingsdaten"(todo Begriffe abstimmen) kommen. For the following experiments we used two datasets: The OpenSubtitle dataset includes move subtitles, so this dataset represents spoken language. On the other hand we have the Reddit Submission Corpus dataset which contains written language. We choose those movie related datasets because we focus all subject about movies and films.

5.1. Original datasets

In diesem Abschnitt befinden sich Informationen zu den rohen Datensets. In the table **??** you can see some general information about the datasets.

OpenSubtitle For each movie there exists a file which contains the movies subtitles in chronologically order. You can see en example in the picture **??**.

	short name	*size [GB]*	*lines [million]*	*data format*	*source*
OpenSubtitle 2016	OpenSubtitle	93	338	XML	[17]
Reddit Submission Corpus 2006-2015	Reddit	885	1'650	JSON	Reddit Comments Corpus [1]

Tabelle 5.1.: Origing and some general information about the Datasets.

5.2. Preprocessing

In diesem Abschnitt wird erklärt, wie wir die Daten für das Training entsprechend extrahieren, filtern und anschliessend die Dialoge zusammengebaut werden.

[1] $https://archive.org/details/2015_reddit_comments_corpus$.

5.2.1. Extraction

Was die erlaubten Zeichen betrifft, so gelten für beide Datensaätze die gleichen folgenden Regelen:

- Gültige Zeichen sind a-z, A-Z, 0-9 und die Satzzeichen .,!?
- Wörter welche ungültige Zeichen beinhalten werden entfernt
- Zwischen Wörter und Satzzeichen wird immer ein Leerzeichen eingefügt

OpenSubtitle Die Daten liegen sortiert nach Film Genre und Jahr vor. In diesen Unterordner befindet sich pro Film eine Datei welche im XML Format den Untertitel beinhaltet. In einem ersten Schritt werden die gezippten Files in den Unterordner gesucht. Anschliessend wird jedes File entzippt und mit Hilfe der XML Library eingelesen. In der Abbildung **??** ist ersichtlich, dass die einzelne Kinder des XML Objekts die Wörter enthalten. Die Wörter werden ausgelesen und entsprechend zu einem Satz kombiniert. Im der Abbildung **??** sehen Sie den oben gezeigten Beispielsatz nach dem Preprocessing. Ein Dialog wiederum wir aus zwei nacheinander folgenden Sätzen erstellt.

Reddit Submission Corpus 2006-2015 Die Reddit Daten sind nach Jahr und Monat sortiert. Pro Monat gibt es eine Datei mit chronologisch aufsteigend sortierten JSON Objekte. Als erstes werden, abhängig von den Programmierter, die Dateien pro gewünschtes Jahr zusammengesucht. Anschliessend werden diese Dateien aufsteigend mit Hilfe der JSON Library ausgelesen und nach entsprechenden Subreddits gefiltert. Ein Beispieleintrag finden Sie in der Abbildung **??**. Da wir ein Dialog System mit Fokus auf Filme erstellen möchten, filterten wir die Daten entsprechend nach den Subreddit Tags, Movies, Films und Television. Nach dem Filtern der Daten, muss die Ursprüngliche Struktur der Kommentare wiederhergestellt werden. Für diesen Vorgang sind die Tags, $Parent_id$, $Link_id$ und Name relevant. Auf der obersten Kommentarebene ist die $Parent_id$ gleich der $Link_id$. Wobei beim ersten Vorkommnis pro ID ein neuer Beitrag beginnt. Für die Kinder eines Kommentars verhält es sich so, dass deren $Parent_id$ gleich dem Name Tag des Elternkommentar ist. Mit Hilfe der Shelve Library werden so die Urpsrünglichen Kommentarstrukturen erstellt. Dialoge werden generiert, indem der Elternknoten jeweils mit allen direkten Kindern kombiniert wird. In der Abbildung **??** sind 2 Beispielkonversationen ersichtlich. Zwischen den Konversationen wird ein Trennzeichen ('¡¡¡¡¡END-CONV¿¿¿¿¿') eingefügt, welches in den Experimenten dem Modell entsprechend das Ende eines Dialogs anzeigt. Todo: Erklärendes Bild einfügen wie zusammengesetz wird, und referenz paper die das auch so zusammengebaut haben

Pro Datensatz erhalten wir nach dem Preprocessing eine Datei, welche pro Zeile eine Aussage enthält. todo: Clique Bildung in text einbauen? entscheid für subreddit tags Visualisierung der CLique gemäss BigQuery todo: visualisierung bildung der dialoge reddit?

5.3. Generate Vocabulary and a train, valid and text Dataset

Da wir die Wörter nicht normalisieren können, müssen wir die zur Verfügung stehenden Wörter begrenzen. Ansonsten würde bei der Softmax Berechnung zu wenig Speicher zur Verfügung stehen(Genauer erklären, referenz auf Kapitel wo erklärt wird). Dementsprechend wird das Vokabular spezifisch für jeden Datensatz generiert. Dabei werden zuerst alle vorkommenden Wörter gezählt und nach der Häufigkeit absteigend ausgegeben. Wir extrahierten daraus 3 Vokabulare, mit den jeweils 25k, 50k und 100k häufigsten Wörter. Die anschliessende Analyse zeigt die Abdeckung des Datensatzes pro Vokabular. Die Ergebnisse sind in der Tabelle **??** ersichtlich.

Wir sehen also, dass trotz der grossen Anzahl Wörter (Angabe Wörter insgesamt pro Dataset), können bereits mit 50k Wörter % der Wörter abgedeckt werden. Ebenfalls Analyse Wieviele Wörter pro Satz Prozentual todo: Einfügen Tabelle Abdeckung Vocab =¿ Korpus todo:zahlen in Text einfliessen lassen einfügen und pickl generierung?

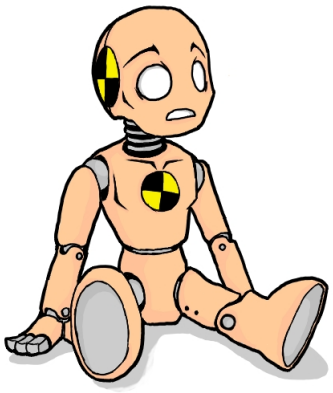


Abbildung 5.1.: Absolute Verteilung der Sätze pro Zeitabstand. [2]

Wir untersuchten die Daten zusätzliche nach Häufigkeiten von Bi- und Trigrammen. Diese Vorgang war sehr rechenintensiv und dauerte entsprechend lange. Die Resultate sind pro Datensatz in den Abbildungen **??** und **??** ersichtlich. Das Ziel dieser Analyse ist es, die später generierten Sätze ebenfalls auf Bi- und Trigramme zu untersuchen um eine mögliche Korrelation auf der Ebene Phrase zu finden. Todo: Wolkenbildung...

Schlussendlich teilten wir die Datensätze auf wobei 97% als Trainingsdaten, 2% als Validierungsdaten und 1% als Testdaten verwendet werden. Die genauen Zahlen befinden sich in der Tabelle 5.2.

	set	*Percent from total [%]*	*size [MB]*	*lines [thousend]*
OpenSubtitle	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

Tabelle 5.2.: In this table you can see the resulting 3 sets per Dataset....

	vocab size [thousand]	*total amount words [%]*	*single coverage [%]*	*0 missing words per utterance [%]*	*1 missing words per utterance [%]*	*2 missing words per utterance [%]*	*total coverage [%]*
OpenSubtitle	25	-	-	-	-	-	-
	50	-	-	-	-	-	-
	100	-	-	-	-	-	-
Reddit	25	-	-	-	-	-	-
	50	-	-	-	-	-	-
	100	-	-	-	-	-	-

Tabelle 5.3.: single coverage abdeckung einzelner wörter. sollte prfübar sein, total words amount mit covab grösse kombiniert.

Beim Durchsichten der Daten, empfanden wir die Qualität des OpenSubtitle Datensatzes als eher schlecht. Wir vermuteten die Ursache in der gesprochenen Sprache, da das Bild als wichtiger Informationsträger fehlt. Eine zusätzliche mögliche Begründung wäre es, dass die Bildszene endet und mit einigem zeitlichen Abstand erst wieder gesprochen wird. Somit würden Sätze kombiniert werden, welche nicht den gleichen Kontext besitzen. Deshalb haben wir die zeitlichen Abstände zwischen den Sätzen analysiert. Die Ergebnisse sind in der Grafik **??** ersichtlich. Aufgrund der Ergebnisse, dass über 80% der Aussagen in einem Zeitabstand kleiner gleich 5 Sekunden aufeinander folgen, verwerfen wir vorerst diese These als Ursache. In der Abbildung **??** sind die Werte grösser 30% auffällig hoch. Dies liegt einerseits an den Übergängen zwischen 2 Datensets und weil alle restlichen Werte in dieser Klasse landen. Dies sehen wir aber nicht als Problem an, weil über 80% der Daten diese Problematik nicht betrifft. Wir unternehmen vorerst keine weiteren

todo: split erklären, dass trainingsbeispiele zusammenbleiben

Preprocessing Regex, Filter bei Reddit 2 Varianten, nach Subrreddit, 30 Wordcount

Generiereung Vokabular und Abdeckung pro Vokabular und corpus

Bi-Gram Tri-Gram Analyse entweder mit WOlke, oder heat mat, tree map Vielleicht gibt es noch eine andere Variante um dies darzustellen (besser), wenn möglich mit paper referenz $http://infosthetics.com/archives/2006/03/subject_tag_news_heat_map.html$ $https://www.quora.com/What-are-alternatives-to-tag-clouds-for-information-visualization$

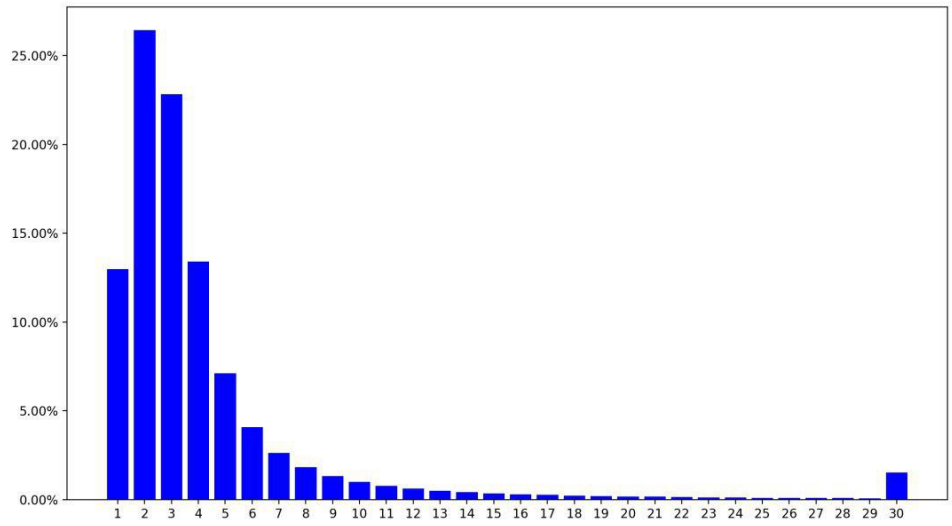


Abbildung 5.2.: Relative Verteilung der berechneten Zeiten (Diskret) in ganzen Sekunden zwischen zwei Äusserungen. Wobei die grössten Anteile die folgende 5 Zeitabstände haben: 13.0% 1 Sekunde, 26.4% 2 Sekunden, 22.8% 3 Sekunden, 13.4% 4 Sekunden und 7% 5 Sekunden. Die resultierende Abdeckung dieser 5 häufigsten Zeitabstände beträgt 82.6%. [3]

Begriffe: Utterance (eine Aussage) Sample (in- utterance, und out utterance)

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. We are using 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 [25], as we are trying to replicate the results from there. 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 CPU 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 [25], 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 bigger 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 2, from a hidden state size of 1024 to 2048, without the need to

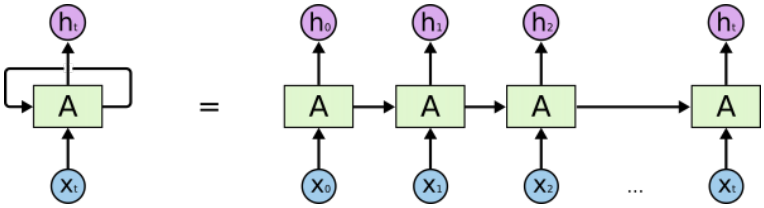


Abbildung 6.1.: Image for illustrating the process of unrolling an RNN over a fixed size of time steps.[2]

sacrifice the size of the vocabulary used (see chapter 6.2 for more informations 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 [15] 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 . On inference time, we then have to use the full softmax layer again to generate the predictions.

Describe more!

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.

6.2. Hyperparameters

Model In summary, for our model we are using the following hyperparameters:

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

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

Tabelle 6.1.: Hyperparameter, which were used for our seq2seq model.

Optimizer As the optimizer, we used *AdaGrad* [8] as in [25] 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 [20].

Training We trained our models on the hardware described in chapter 4.6 with the software packages from chapter 4.7. As this model take a really long time to converge (in [25], the authors trained their model for several weeks), we also had to train ours for a long time. To show the development of the model throughout this time, we took several snapshots, approximately every second day or about after every 500'000 batches. We are going to use them in the analysis lying ahead.

6.3. Evaluation

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7. Experiments

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

* 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

7.1. Analysis of Learning Process

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.2. Generated outputs over time

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift –

mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.3. Comparison with CleverBot

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.4. N-Gram Analysis

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.5. Sent2Vec Analysis

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene

Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

7.6. Internal Embedding for Input Sequences

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

8. Conclusion

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

9. Future Work

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

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.

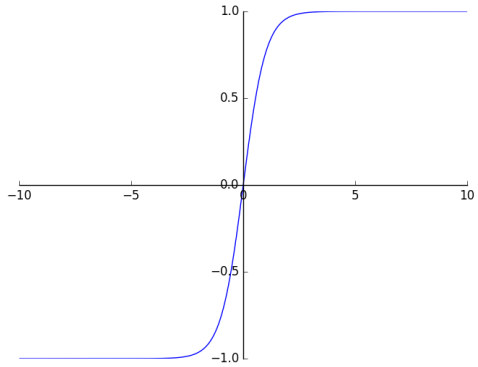
These neurons accept n input values $\mathbf{x} = (x_0, x_1, \ldots, x_n)$ and use them to compute a single output value o. A unique weight w_n from the set $\mathbf{w} = \{w_0, w_1, \ldots, w_n\}$ is assigned to each of the input values. The input value of x_0 is almost always set to 1 and not further 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:

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

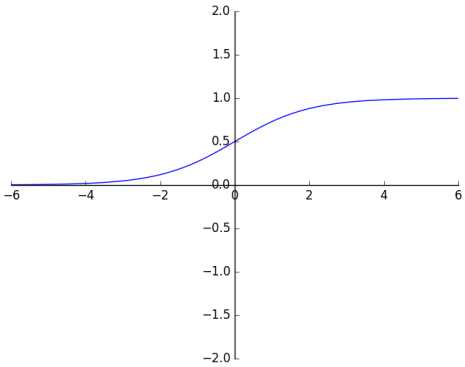
The activation function φ is responsible for squeezing the result of the computation of a neuron into a predefined range of values; for example using *tanh* always results in output values in the range $[-1, +1]$, no matter which scale the input values originally had. Examples of commonly used activation 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*.

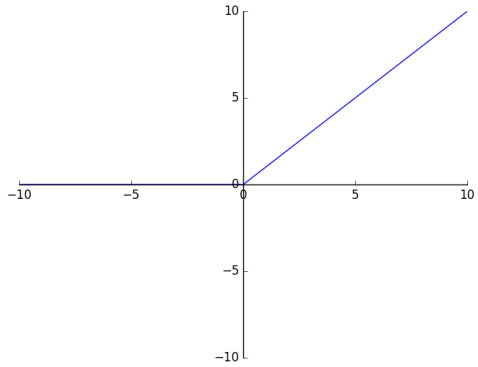
The input data $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ is enters the NN through the input layer. At this



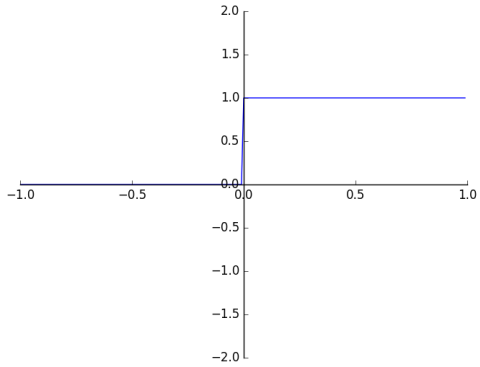
(a) $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$



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



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



(d) blub

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

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 to.

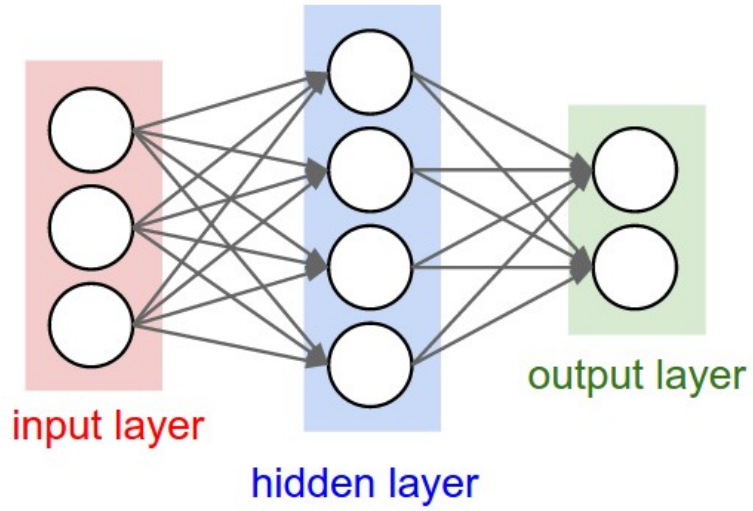


Abbildung A.2.: Simplified visualization of a NN with one input, hidden and output layer.

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:

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

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 $\mathbf{w}$:

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

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* [8] or *AdaDelta* [29], 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 [21].

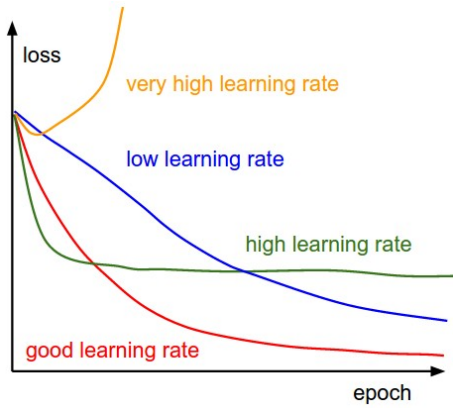


Abbildung A.3.: Visualization of the development of a loss function when using different learning rates.[1]

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

B. Additional Charts

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

C. Using The Software System

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

Glossary

.

.

GPU Graphical Processing Unit.

NN Neural Network.

.

Tabellenverzeichnis

Abbildungsverzeichnis

References

[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 und X. Zheng, *Tensorflow: Large-scale machine learning on heterogeneous systems*, Software available from tensorflow.org, 2015. Adresse: `http://tensorflow.org/`.

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

[3] K. Cho, A. Courville und Y. Bengio, „Describing multimedia content using attention-based encoder-decoder networks“, *IEEE Transactions on Multimedia*, Bd. 17, Nr. 11, S. 1875–1886, 2015.

[4] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk und Y. Bengio, „Learning phrase representations using rnn encoder-decoder for statistical machine translation“, *ArXiv preprint arXiv:1406.1078*, S. 1724–1734, 2014.

[5] J. Chung, C. Gulcehre, K. Cho und Y. Bengio, „Empirical evaluation of gated recurrent neural networks on sequence modeling“, *ArXiv preprint arXiv:1412.3555*, 2014.

[6] R. Collobert, K. Kavukcuoglu und C. Farabet, „Torch7: A matlab-like environment for machine learning“, in *BigLearn, NIPS Workshop*, 2011.

[7] R. Desimone und J. Duncan, „Neural mechanisms of selective visual attention“, *Annual review of neuroscience*, Bd. 18, Nr. 1, S. 193–222, 1995.

[8] J. Duchi, E. Hazan und Y. Singer, „Adaptive subgradient methods for online learning and stochastic optimization“, *Journal of Machine Learning Research*, Bd. 12, Nr. Jul, S. 2121–2159, 2011.

[9] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink und J. Schmidhuber, „Lstm: A search space odyssey“, *IEEE transactions on neural networks and learning systems*, 2016.

[10] K. Gregor, I. Danihelka, A. Graves, D. J. Rezende und D. Wierstra, „Draw: A recurrent neural network for image generation“, *ArXiv preprint arXiv:1502.04623*, 2015.

[11] D. von Grünigen, M. Weilenmann, J. Deriu und M. Cieliebak, „Potential and limitations of cross-domain sentiment classification“, *SocialNLP 2017*, S. 17–25, 2017.

[12] S. Hochreiter, „Untersuchungen zu dynamischen neuronalen netzen“, Diss., diploma thesis, institut für informatik, lehrstuhl prof. brauer, technische universität münchen, 1991.

[13] S. Hochreiter und J. Schmidhuber, „Long short-term memory“, *Neural computation*, Bd. 9, Nr. 8, S. 1735–1780, 1997.

[14] L. Itti, C. Koch und E. Niebur, „A model of saliency-based visual attention for rapid scene analysis“, *IEEE Transactions on pattern analysis and machine intelligence*, Bd. 20, Nr. 11, S. 1254–1259, 1998.

[15] S. Jean, K. Cho, R. Memisevic und Y. Bengio, „On using very large target vocabulary for neural machine translation“, *CoRR*, Bd. abs/1412.2007, 2014. Adresse: `http://arxiv.org/abs/1412.2007`.

[16] N. Kalchbrenner und P. Blunsom, „Recurrent continuous translation models.“, in *EMNLP*, Bd. 3, 2013, S. 413–422.

[17] P. Lison und 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.

[18] V. Mnih, N. Heess, A. Graves u. a., „Recurrent models of visual attention“, in *Advances in neural information processing systems*, 2014, S. 2204–2212.

[19] M. Pagliardini, P. Gupta und M. Jaggi, „Unsupervised learning of sentence embeddings using compositional n-gram features“, *ArXiv*, 2017. arXiv: `1703.02507 [cs.CL]`.

[20] R. Pascanu, T. Mikolov und Y. Bengio, „On the difficulty of training recurrent neural networks.“, *ICML (3)*, Bd. 28, S. 1310–1318, 2013.

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

[22] H. T. Siegelmann und E. D. Sontag, „On the computational power of neural nets“, *Journal of computer and system sciences*, Bd. 50, Nr. 1, S. 132–150, 1995.

[23] I. Sutskever, O. Vinyals und Q. V. Le, „Sequence to sequence learning with neural networks“, in *Advances in neural information processing systems*, 2014, S. 3104–3112.

[24] Theano Development Team, „Theano: a Python framework for fast computation of mathematical expressions“, *ArXiv e-prints*, Bd. abs/1605.02688, Mai 2016. Adresse: `http://arxiv.org/abs/1605.02688`.

[25] O. Vinyals und Q. Le, „A neural conversational model“, *ArXiv preprint arXiv:1506.05869*, 2015.

[26] P. J. Werbos, „Backpropagation through time: What it does and how to do it“, 10, Bd. 78, IEEE, 1990, S. 1550–1560.

[27] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong und W.-c. Woo, „Convolutional lstm network: A machine learning approach for precipitation nowcasting“, in *Advances in Neural Information Processing Systems*, 2015, S. 802–810.

[28] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel und Y. Bengio, „Show, attend and tell: Neural image caption generation with visual attention“, in *International Conference on Machine Learning*, 2015, S. 2048–2057.

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