M.Sc. Thesis
Master of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

Language modelling using deep learning

Generating answers to medical questions using recurrent neural networks

Lasse Regin Nielsen (s123815@student.dtu.dk)

Kongens Lyngby 2017

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 303B
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Summary

This thesis investigates the application of Long-short Term Memory-based Recurrent Neural Networks (RNN) to the Short Text Conversation problem. Specifically the task of generating answers to medical questions is studied.

Training data is gathered from online services containing user-generated questions with associated answers generated by professionals. The proposed models are trained and evaluated using data originating from different online services like WebMD, HealthTap, and iCliniq, from which an optimal dataset is determined.

The models proposed to solving the task are mainly inspired by models used in Neural Machine Translation but also contains extensions based on Transfer learning and Multi-task learning, are all based on the Encoder-decoder framework having an encoder compute a latent vector representation of a question from which the state of a decoder is initialized and used to generate an answer, and are all trained end-to-end.

One model architecture containing a decoder RNN with two "*modes*", controlled by a binary input to the decoder, are proposed. One mode being a "*language-model*" mode where the decoder is trained on general medical/health-related text, and another being a "*answer generating*" mode where the decoder learns to generate answers to given encoded questions.

Another model architecture handling the answer generation task and the task of classifying question categories as a combined task is propsed, where the network classifies the question category from the final state of the encoder, and feeds the predicted class as an extra input to the decoder.

Finally a separate RNN-based language model is trained on general medical/health-related text and used to assist the proposed models during inference, by merging their probabilities at each time-step, in an attempt on improving the generated answer quality.

Summary (danish)

Dette projekt undersøger anvendelsen af Long-short Term Memory-baserede Recurrent Neurale Netværk (RNN) på kort tekst samtale problemet. Mere specifikt så er automatisk generering af svar til medicinske spørgsmål undersøgt.

Trænings data er samlet fra online servicer indeholdende bruger-genererede spørgsmål med dertilhørende svar givet af professionelle. De foreslåede modeller er trænet og evalueret på data fra forskellige online servicer såsom WebMD, HealthTap og iCliniq, hvorfra det optimale dataset er bestemt.

De foreslåede modeller til løsningen af problemet er hovedsagligt inspireret af modeller anvendt i Neural Maskinoversættelse men indeholder også udvidelser baseret på Transfer learning og Multi-task learning, er alle baseret på Encoder-decoder frameworket hvor en encoder projekterer et spørgsmål til en latent vektor representation, hvorfra en decoder benytter vektor representationen til genereringen af et svar, og er alle trænet end-to-end.

En model arkitektur indeholdende en decoder RNN med to tilstande, kontrolleret af et binært input til decoderen, er foreslået. Èn tilstand værende en "*sprog model*" tilstand hvor decoderen er trænet på generelt medicinsk/helbreds-relateret tekst, og en anden værende en "*svar genererende*" tilstand hvor decoderen lærer at generere svar til givne spørgsmål.

En anden model arkitektur der behandler svar genereringen og klassificeringen af et spørgsmåls kategori som èn samlet opgave, er foreslået. Netværket klassificerer spørgsmålets kategori baseret på den endelige tilstand af encodered, og fodrer den predikterede klasse som ekstra input til decoderen.

Endelig er en separat RNN-baseret sprog model trænet på generelt medicinsk/helbreds-relateret tekst and anvendt til at assistere de foreslåede modeller under inferering, ved at sammenlægge deres sandsynligheder ved hvert tids skridt i et forsøg på at forbedre kvaliteten af de genererede svar.

Preface

This thesis was prepared at the department of Applied Mathematics and Computer Science at the Technical University of Denmark in cooperation with FindZebra in fulfilment of the requirements for acquiring a M.Sc. in Mathematical Modelling and Computation. The code developed throughout this project is publicly avilable at Github [1].

Kongens Lyngby, June 27, 2017

Lasse Regin Nielsen (s123815@student.dtu.dk)

[1] https://github.com/LasseRegin/master-thesis-deep-learning

Acknowledgements

I would like to thank my supervisor Professor Ole Winther at DTU Compute for his guidance and for helping me putting this project together.

Furthermore I would like to thank my co-supervisors PhD student Dan Tito Svenstrup and Jonas Hansen from FindZebra for all their help and for always being available throughout the project process.

Contents

CHAPTER 1

Introduction

Deep Learning or specifically Deep Neural Networks (DNN's) have shown great performance within many topics such as object detection and classification in images using Convolutional Neural Networks (CNN's) [33, 27], recognising speech using Recurrent Neural Networks (RNN's) [3, 1], and named entity recognition also using RNN's [21].

Other topics such as Statistical Machine Translation (SMT) have been found more difficult to obtain great performance in using DNN's, but recent research have shown very promising results using RNN's. Specifically the end-to-end approach referred to as "*Neural Machine Translation*" (NMT) [2, 6], which uses the generic encoder-decoder framework proposed in [32] encoding sequences to fixed-sized latent vector representations from which a decoder generates a new sequence (see Figure 1.1), have been shown to beat state-of-the-art phrase-based translation systems.

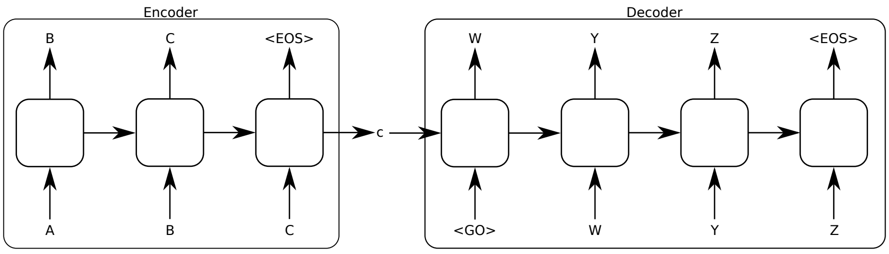


Figure 1.1: Visualization of the encoder-decoder framework with c being the fixed-size latent vector representation of the input sequence `ABC` and the generated sequence being `WYZ`.

Another difficult problem related to the SMT problem, which still havn't been solved using Deep Learning, is the so-called "*Short Text Conversation*" (STC) problem. STC considers the generation of a suitable human-like answer/response to a single given (relatively short) statement, post, or question. One can say it is a sub-problem of Conversational modelling [34] which is the task solving e.g. when creating chatbots.

In this project the STC problem is considered limitted within the topics of health and medical questions/answers (Q/A's). I.e. the problem of generating answers to given medical questions is studied.

1.1 Related work

The STC problem have previously been attempted solved using Information Retrieval (IR) approaches, SMT approaches, and also a more generic end-to-end approach using an encoder-decoder framework.

In [17] the STC problem is attempted solved using an IR approach. Generally IR tries to extract appropriate responses from a given set of fixed post-response pairs. The approach in [17] consists of the following 3 steps:

1. Retrieve query (the post)
2. Find matching post-response pairs from an index of post-response pairs
3. Rank the matching post-response pairs and return the highest ranked pair

where the ranking is learned using a linear Ranking SVM, and the index of post-response pairs is created using data collected from Twitter. An obvious limitation of the IR approach is the use of a fixed set of responses, i.e. it is not possible to create a response which doesn't already exist in the dataset. Also the IR approach was found to require a set of manually designed matching features in order to find proper post-response matches from the query, which is something you would like to avoid.

The first to investigate the use of the SMT approach for generating conversational responses used the open source phrase-based decoder "*Moses*" [12] trained on 1.3 million post-response pairs collected from Twitter[1], where each response to a post together with the post itself make up a single post-response pair [28]. It is shown that the SMT approach outperforms information-retrieval approaches, and that the responses generated by the SMT approach is preferred over human responses in 15% of the cases.
Some of the challenges with generating conversational responses learned from post-response data vs. generating translations from bilingual data (the task in SMT) are also identified in [28], and they include the following two essential challenges:

1. Conversational post-response pairs are not semantically equivalent where bilingual translation pairs usually are, which results in the set of possible valid responses to be much larger for post-response pairs.
2. Bilingual translation pairs usually have many words aligned between the pair whereas post-response pairs often don't have any aligned words; sometimes they might even have no words in common.

[1] Only the first two responses to a post were used and Twitter posts and responses are all limited to 140 characters.

This suggests that the STC problem is a difficult task to solve using similar approaches as used in SMT.

In [30] a so-called “*Neural Responding Machine*” (NRM) is proposed for generating responses to posts from the Twitter-like website Weibo using the encoder-decoder framework with RNN’s used for the encoder and decoder. The model is trained on approximately 4.5 million post-response pairs collected from Weibo where again each response to a post, together with the post itself, make up a post-response pair. The NRM is shown to outperform both retrieval-based and SMT-based models like the one used in [12].
The models proposed in this project for solving the STC problem is based on the approaches used in [30], but also proposes extensions based on Transfer learning and Mulit-task learning which will be explained in Section 2.13.

In this project a RNN-based language model is trained on general text from medical/health-related wikipedia articles and papers, which is then utilized to “*help*” an answer-generating model to generate better answers.

1.2 Problem motivation

The problem considered in this project is the lack of available answers to any given medical/health-related question. The internet is full of forums allowing people to ask these questions, from which some of them retrieves great answers to. However browsing these questions to locate one similar to your own, and also finding one with a satisfying answer is a hard and time-consuming task, and posting the question yourself is even worse.
By automating the process of generating answers to these questions, and thereby creating a kind of “*digital doctor*” one could solve this problem.
The problem thereby limits to the STC problem, i.e. given a medical/health-related question a single suitable answer should be generated.

1.3 The goal

Based on the challenges listed in [28] about the STC problem, and the lack of available Q/A data within medical/health topics compared to data sources for general conversations such as Twitter and Weibo, creating an end-to-end well-performing “*digital doctor*” is obviously very optimistic. However since the amount of available general medical/health text from e.g. Wikipedia etc. is very big, hence generating somewhat useful answers for a limited set of questions seems possible.
Thus the goal of this project is to train an end-to-end model using RNN’s and the encoder-decoder framework, which should be able to generate sensible, somewhat useful answers to a small set of given medical/health questions.

The data used for training will be scraped from websites containing medical Q/A's where answers have been given by professionals.

1.4 Report structure

The report is structured in the following way:

- **Introduction**: Introduction to and motivation of the problem, previous related and useful work, and the goal of project.
- **Theory**: General theory of the different fields, methods, and, models used in the project including neural networks, how they are trained, regularized, and evaluated, and their extensions.
- **Methods**: Details about implementation, data, proposed models, and training techniques.
- **Results and discussion**: Results from the performed experiments including training stability, performance, and comparison of models are shown and discussed.
- **Conclusion**: The conclusions that can be drawn from the results are listed.

1.5 Mathematical notation

The following mathematical notation is used throughout the report

Scalars are simply lowercase letters	a
Vectors are denoted lowercase bold letters and are by default column vectors	$\mathbf{a}$
Matrices are denoted capital letters	A

CHAPTER 2 Theory

2.1 Linear regression

The linear model is one of the most fundamental statistical models, which is applied in many areas such as statistics and machine learning, hence the importance of the model. Assuming a linear relationship between a set of input vectors $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ with $\mathbf{x}_i \in \mathbb{R}^p$ and output vectors $\{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_N\}$ with $\mathbf{y}_i \in \mathbb{R}^M$ the linear model is defined as

$$\mathbf{y}_i = \sum_{j=1}^{p} \mathbf{w}_j x_{i,j} + \mathbf{b} + \boldsymbol{\varepsilon}_i \quad \text{for} \quad i = 1 \ldots N \tag{2.1}$$

with $\{\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_p\}$ being a weight vector of size $\mathbf{w}_j \in \mathbb{R}^M$ for each dimension p, and $\mathbf{b} \in \mathbb{R}^M$ being the bias i.e. the "*learned*" parameters of the model, and $\varepsilon_i \sim \mathcal{N}(0, \sigma_\varepsilon^2) \in \mathbb{R}^M$ being the noise in the data which is assumed to follow a gaussian distribution with zero mean and a standard deviation of σ_ε^2 [10]. If this assumption does not hold, the linear model should not be used.

Letting $\mathbf{w}_0 = \{1\}^p$ and $\tilde{\mathbf{x}}_i = \begin{bmatrix} 1 \\ \mathbf{x}_i \end{bmatrix}$ $\forall i$ the bias is incorporated in the weight matrices and the model definition reduces to

$$\mathbf{y}_i = \sum_{j=0}^{p} \mathbf{w}_j \tilde{x}_{i,j} + \boldsymbol{\varepsilon}_i \quad \text{for} \quad i = 1 \ldots N \tag{2.2}$$

By denoting

$$\mathrm{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \ldots & \tilde{\mathbf{x}}_N \end{bmatrix} \in \mathbb{R}^{(p+1)\times N} \tag{2.3}$$

$$\mathrm{Y} = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \ldots & \mathbf{y}_N \end{bmatrix} \in \mathbb{R}^{M\times N} \tag{2.4}$$

$$\mathrm{W} = \begin{bmatrix} \mathbf{w}_0 & \mathbf{w}_1 & \ldots & \mathbf{w}_p \end{bmatrix} \in \mathbb{R}^{M\times (p+1)} \tag{2.5}$$

$$\mathrm{E} = \begin{bmatrix} \boldsymbol{\varepsilon}_1 & \boldsymbol{\varepsilon}_2 & \ldots & \boldsymbol{\varepsilon}_N \end{bmatrix} \in \mathbb{R}^{M\times N} \tag{2.6}$$

the linear model can be expressed in the following compact matrix notation

$$\mathrm{Y} = \mathrm{W}\mathrm{X} + \mathrm{E}. \tag{2.7}$$

Determining the values of the weight matrix W can be done in multiple ways, but the most common way is to minimize the "*Residual Sum-of-Squares*" (RSS), which is given as

$$\begin{aligned}
\mathrm{RSS}(\mathrm{W}) &= \sum_{i=i}^{N}\sum_{j=0}^{M}\left(y_{i,j} - \sum_{k=0}^{p} w_{k,j}\tilde{x}_{i,k}\right)^2 && (2.8)\\
&= \sum_{i=i}^{N}\sum_{j=0}^{M}(\mathbf{y}_i - \mathrm{W}\tilde{\mathbf{x}}_i)_j^2 && (2.9)\\
&= \sum_{i=i}^{N}\left((\mathrm{Y}-\mathrm{WX})^T(\mathrm{Y}-\mathrm{WX})\mathbf{1}\right)_i && (2.10)\\
&= \mathbf{1}^T(\mathrm{Y}-\mathrm{WX})^T(\mathrm{Y}-\mathrm{WX})\mathbf{1}. && (2.11)
\end{aligned}$$

Since the expression is quadratic in the parameters a unique solution can be found analytically. The RSS-estimate of the weight matrix is therefore given by

$$\hat{\mathrm{W}}_{\mathrm{RSS}} = \underset{\mathrm{W}}{\mathrm{argmin}} \quad \mathrm{RSS}(\mathrm{W}) \tag{2.12}$$

which can be found by solving

$$\frac{\partial \mathrm{RSS}(\mathrm{W})}{\partial \mathrm{W}} = 0 \tag{2.13}$$

which leads to the closed form solution (see derivation in Appendix A.1)

$$\hat{\mathrm{W}}_{\mathrm{RSS}} = \mathrm{YX}^T\left(\mathrm{XX}^T\right)^{-1} \tag{2.14}$$

2.2 Artifical neural networks

"*Artifical neural networks*" (ANN) or just "*Neural networks*" (NN) is a class of pattern recognition models inspired by the human brain, which have taken up a big part of research within machine learning, and due to it's popularity in wide applications have obtained it's own branch of machine learning called "*Deep Learning*".
A neural network consists of connected nodes, where the nodes resembles biological neurons in the human brain and the connections resembles the axons.
Where the biological neuron receives an input signal from one or multiple axons and outputs a certain activation signal, so does the node in the neural network.
From now on and throughout the report the nodes in an ANN will be refered to as a "*neuron*".
Given a set of input signals $\{x_1, x_2, \ldots, x_N\}$ and a resulting output activation signal a the neuron can be visualized as in Figure 2.1

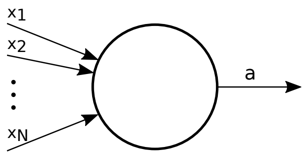


Figure 2.1: Visualization of a single linear neural network neuron with N input connections.

where the activation a will be a weighted sum of the inputs

$$a = \sum_{i=1}^{N} x_i w_i \tag{2.15}$$

i.e. each neuron will act as a linear regression model presented in Section 2.1, which then will act as the input to another neuron.

In order to fit nonlinear patterns in the data, and since stacking multiple linear regression models will result in linear regression itself, the activation is usually transformed using a nonlinear function known as an "*activation function*" denoted $\sigma(\cdot)$, i.e. the new output of each neuron will be

$$z = \sigma(a) = \sigma\left(\sum_{i=1}^{N} x_i w_i\right). \tag{2.16}$$

Common choices of activation functions are so-called "*sigmoidal*" functions which include (but is not limited to) the sigmoid function and the hyperbolic tangent

$$\text{Sigmoid}: \qquad \sigma(x) = \frac{1}{1+e^{-x}} \tag{2.17}$$

$$\text{Hyperbolic tangent}: \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{2.18}$$

which both have the sigmoidal form seen in Figure 2.2 but differs in the output range where the sigmoid function maps an input to the range $[0,1]$, and corresponds to each neuron being a logistic regression model, and the hyperbolic tangent maps to the range $[-1,1]$.

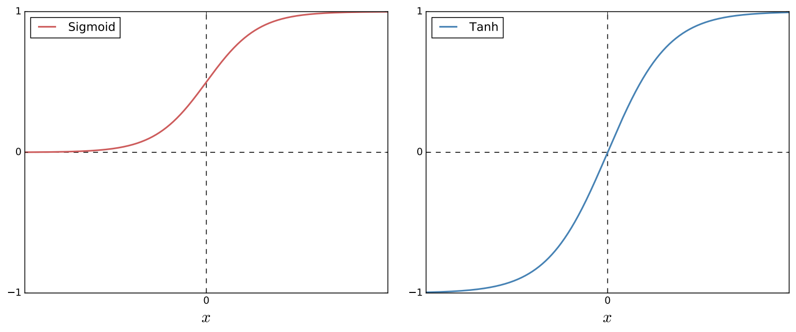


Figure 2.2: Sigmoid vs. hyperbolic tangent.

From some algebraic manipulation (see Appendix A.2) it is found that the two functions have the following relation

$$\tanh(x) = 2\sigma(2x) - 1 \tag{2.19}$$

which makes it difficult to argue choosing one over the other as activation function, since the model should be able to learn the same patterns using any of the two. Other activation functions includes "*Rectified linear unit*" (ReLU) given as

$$\mathrm{ReLU}(x) = \max(0, x), \tag{2.20}$$

"*Leaky rectified linear unit*" (Leaky ReLU), the sign(·) function, and many more. The only restriction to activation functions is that they must be differentiable making it possible to compute its gradient w.r.t. a given set of weights. In this project the tanh, sigmoid, and the ReLU function are all used for different purposes. This will be further described later.
In order to distinguish the type of neurons used in a network, a neuron using a "*sigmoidal*" activation function will from now on be visualized as in Figure 2.3.

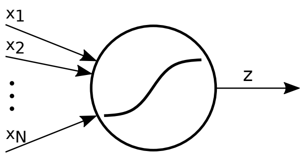


Figure 2.3: Visualization of a single neural network neuron with "*sigmoidal*" activation function.

These neurons can be aligned, connected, and stacked in different ways constructing different type of networks often refered to as "*network architectures*".
The following sections will explain the following network types

- Feed-forward neural network

- Recurrent neural network
- Bidirectional recurrent neural network

2.2.1 Feed-forward neural network

The Feed-Forward Neural Network (FFNN) is one of the simplest types of neural networks where, as the name indicates, information is only fed forward in the network. FFNN's consists of an "*input layer*", and one or more so-called "*hidden layers*" with the last hidden layer being the "*output layer*". Despite the simplicity of the FFNN it is a very powerful model. This is emphasized in [14] where it is shown that a single layer FFNN is capable of approximating any function.
In Figure 2.4 a 3-layer FFNN is shown with an input of size 3 and an output of size 2.

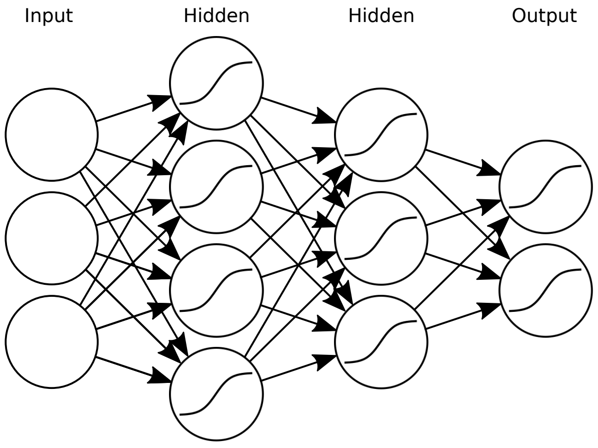


Figure 2.4: Visualization of a 3-layer feed-forward neural network with input $\mathbf{x} \in \mathbb{R}^3$ and output $\mathbf{y} \in \mathbb{R}^2$.

Denoting L the number of hidden layers and $N^{(l)}$ the number of neurons in layer l, with $l = 0$ being the input layer and $l = L$ being the output layer, and denoting the weight matrix between layer $l-1$ and layer l as $\mathrm{W}^{(l)} \in \mathbb{R}^{N^{(l)} \times N^{(l-1)}}$ constructed by concatenating the weight-vectors associated with each neuron in layer $l-1$, i.e. $\mathrm{W}^{(l)} = \begin{bmatrix} \mathbf{w}_1^{(l)} & \mathbf{w}_2^{(l)} & \dots & \mathbf{w}_{N^{(l-1)}}^{(l)} \end{bmatrix}$ one can compute the activation of neuron k in layer l like in (2.16) using

$$z_k^{(l)} = \sigma\left(a_k^{(l)}\right) = \sigma\left(\sum_{k'=1}^{N^{(l-1)}} z_{k'}^{(l-1)} w_{k,k'}^{(l)}\right) \tag{2.21}$$

with $z^{(0)} = \mathbf{x} \in \mathbb{R}^p$ being the input vector, and thereby $z^{(L)} = \hat{\mathbf{y}} \in \mathbb{R}^{N^{(L)}}$ the network output with the number of output neurons $N^{(L)}$ being the dimensionality of the output vector $\hat{\mathbf{y}}$. For a classification problem with K classes the number of output neurons $N^{(L)}$ is usually $N^{(L)} = K$ where each neuron k represents the probability of class C_k denoted $p\left(C_k|\mathbf{x}\right)$ computed using the "*softmax*" function given as

$$p\left(C_k|\mathbf{x}\right) = z_k^{(L)} = \frac{e^{a_k^{(L)}}}{\sum_{k'=1}^{N^{(L)}} e^{a_{k'}^{(L)}}} \tag{2.22}$$

which acts as generalization of the sigmoid function for multidimensional input, and due to it's normalization factor can be directly treated as a probability, since

$$p\left(C_k|\mathbf{x}\right) \in [0,1]\ \forall k \quad \text{and} \quad \sum_{k=1}^{N^{(L)}} p\left(C_k|\mathbf{x}\right) = 1. \tag{2.23}$$

During training the target vector $\mathbf{y}$ for each category k can be represented using the "1-*of*-*K*" coding scheme, where all elements of $\mathbf{y}$ are 0 except element k which is a 1. The number of hidden layers L indicates the "*depth*" of the network, where "*deep*" networks have the ability to learn abstract feature combinations from the data due to it's complexity, where "*shallow*" networks might not be able.
However the risk of "*overfitting*" the training data, where the network starts to model the noise in the data, increases with the depth of the network. This is usually a trade-off between the amount of training data and the difficulty of the problem trying to solve.
Denoting $\mathcal{W} = \{\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, \ldots, \mathrm{W}^{(L)}\}$ as the set of network parameters, the network function f computing the output for a given input is uniquely defined as

$$\mathbf{z}^{(L)} = \hat{\mathbf{y}} = f(\mathbf{x}|\mathcal{W}), \tag{2.24}$$

i.e. the weights define the network. In order to find the optimal weights for a given network and a given set of training data, one must define a loss function for the network for which the weights should be optimized w.r.t.

2.2.1.1 Loss function

Given a classification problem like in (2.21) with K classes we want to find the weights which maximizes the conditional probability $p\left(C_k|\mathbf{x}\right)$ for the correct classes k, i.e. given a dataset $\mathcal{D} = \{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$ we want to maximize the probability assigned to the dataset by the network

$$\mathcal{W}_{\mathrm{ML}} = \underset{\mathcal{W}}{\operatorname{argmax}} \prod_{i=1}^{N} p\left(\mathbf{y}_i|\mathbf{x}_i, \mathcal{W}\right) \tag{2.25}$$

which leads to a maximum-likelihood estimate of the optimal weights $\mathcal{W}_{\mathrm{ML}}$ [8].

One can then define the loss function $\mathcal{L}(\mathcal{D})$ as the probability in (2.25), or more commanly as the negative natural logarithm of the probability for the given dataset $\mathcal{D}$ as

$$\mathcal{L}(\mathcal{D}) = -\ln \prod_{i=1}^{N} p\left(\mathbf{y}_i | \mathbf{x}_i, \mathcal{W}\right) \tag{2.26}$$

which then must be minimized instead of maximized like in (2.25), hence also more suitable for a "*loss*" function.
Using the rule $\ln(a \cdot b) = \ln(a) + \ln(b)$ it is seen that

$$\mathcal{L}(\mathcal{D}) = -\sum_{i=1}^{N} \ln p\left(\mathbf{y}_i | \mathbf{x}_i, \mathcal{W}\right) \tag{2.27}$$

i.e. we can define an "*example loss*" $\mathcal{L}(\mathbf{x}, \mathbf{y})$ for each example $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ as

$$\mathcal{L}(\mathbf{x}, \mathbf{y}) = -\ln p\left(\mathbf{y} | \mathbf{x}, \mathcal{W}\right) \tag{2.28}$$

and thereby we have

$$\mathcal{L}(\mathcal{D}) = \sum_{i=1}^{N} \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i) \tag{2.29}$$

which we can optimize w.r.t. the network weights in order to obtain an estimate of the optimal weights $\mathcal{W}_{\mathrm{ML}}$ for a given training dataset $\mathcal{D}$ [8].
For the given classification problem with K classes we define the per-example network probability $p\left(\mathbf{y} | \mathbf{x}, \mathcal{W}\right)$ from the conditional probability given in (2.22)

$$p\left(\mathbf{y} | \mathbf{x}, \mathcal{W}\right) = \prod_{k=1}^{K} p\left(C_k | \mathbf{x}\right)^{y_k} \tag{2.30}$$

with $\mathbf{y}$ being the "*1-of-K*" encoded class vector. By substituting (2.30) into (2.28) we get

$$\begin{aligned} \mathcal{L}(\mathbf{x}, \mathbf{y}) &= -\ln \prod_{k=1}^{K} p\left(C_k | \mathbf{x}\right)^{y_k} && (2.31) \\ &= -\sum_{k=1}^{K} y_k \ln p\left(C_k | \mathbf{x}\right) && (2.32) \\ &= -\sum_{k=1}^{K} y_k \ln z_k^{(L)} && (2.33) \end{aligned}$$

which is known as the "*cross-entropy*" loss function. Using the "*cross-entropy*" as loss function the network can be trained using any gradient-based optimization technique (see Section 2.8.1), since the loss function and the network are constructed from differentiable operators. This means in order to minimize the loss function, and thereby find the optimal weights $\mathcal{W}_{\mathrm{ML}}$, the gradient of the loss function w.r.t. the weights $\mathcal{W}$, i.e.

$$\frac{\partial \mathcal{L}(\mathcal{D})}{\partial \mathcal{W}} = \sum_{i=1}^{N} \frac{\partial \mathcal{L}(\mathbf{x}_i, \mathbf{y}_i)}{\partial \mathcal{W}} \tag{2.34}$$

needs to be computed. This can be done using the "*Backpropagation*" method which will be explained in Section 2.2.1.2.

2.2.1.2 Backpropagation

The backpropagation algorithm is an algorithm for efficiently computing the gradient of some given loss function w.r.t. the weights of a FFNN. I.e. given the weights of each layer of a given network

$$\mathcal{W} = \{\mathrm{W}^{(1)}, \mathrm{W}^{(2)}, \ldots, \mathrm{W}^{(L)}\} \tag{2.35}$$

the backpropagation algorithm computes

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial \mathcal{W}} = \left\{ \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial \mathrm{W}^{(1)}}, \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial \mathrm{W}^{(2)}}, \ldots, \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial \mathrm{W}^{(L)}} \right\}. \tag{2.36}$$

Given a training example $(\mathbf{x}, \mathbf{y}) \in \mathcal{D}$ from the dataset $\mathcal{D}$ the first step of the algorithm is to pass the input vector $\mathbf{x}$ through the network, and thereby compute the networks output vector $\hat{\mathbf{y}}$, called the "*Forward pass*", with the current weight values of the network.
Second step is the so-called "*Backward pass*" which involves propagating errors back through the network.
Starting from the output layer one can compute the gradient of the loss function for a single given weight $w_{i,j}^{L}$ from using the chain-rule ($\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$) [5]

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial w_{i,j}^{(L)}} = \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial a_i^{(L)}} \frac{\partial a_i^{(L)}}{\partial w_{i,j}^{(L)}} \tag{2.37}$$

where it is seen from (2.15) that

$$\frac{\partial a_i^{(L)}}{\partial w_{i,j}^{(L)}} = z_j^{(L-1)} \tag{2.38}$$

with $z_j^{(L-1)}$ being the activation of the j'th neuron in layer $L-1$, and we define

$$\delta_i^{(L)} = \frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial a_i^{(L)}} \tag{2.39}$$

as the "*error*" of the i'th output neuron.

This defines the gradient of the loss function w.r.t. the weights leading to the output layer as

$$\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{y})}{\partial w_{i,j}^{(L)}} = \delta_i^{(L)} z_j^{(L-1)}. \tag{2.40}$$

By defining the error $\delta_i^{(l)}$ for $l = 1, \ldots L-1$ for the remaining layers like in (2.39) we have

$$\delta_i^{(l)} = \frac{\partial \mathcal{L}(\mathbf{x},\mathbf{y})}{\partial a_i^{(l)}} = \sum_{k=1}^{N^{(l+1)}} \frac{\partial \mathcal{L}(\mathbf{x},\mathbf{y})}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} \tag{2.41}$$

where it is seen that $\frac{\partial \mathcal{L}(\mathbf{x},\mathbf{y})}{\partial a_k^{(l+1)}} = \delta_k^{(l+1)}$ and using (2.21) the term $a_k^{(l+1)}$ can be expressed as

$$\begin{aligned} a_k^{(l+1)} &= \sum_{k'=1}^{N^{(l)}} z_{k'^{(l)}} w_{k,k'}^{(l+1)} && (2.42)\\ &= \sum_{k'=1}^{N^{(l)}} z_{k'^{(l)}} w_{k,k'}^{(l+1)} && (2.43)\\ &= \sum_{k'=1}^{N^{(l)}} \sigma\left(a_{k'}^{(l)}\right) w_{k,k'}^{(l+1)} && (2.44) \end{aligned}$$

which from inserting into (2.41) yields

$$\delta_i^{(l)} = \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} \frac{\partial \left(\sum_{k'=1}^{N^{(l)}} \sigma\left(a_{k'}^{(l)}\right) w_{k,k'}^{(l+1)} \right)}{\partial a_i^{(l)}} \tag{2.45}$$

from where it is seen that the only term remaining when taking the derivative of the sum w.r.t. $a_i^{(l)}$ is the term where $k' = i$, i.e. the derivative simplifies to

$$\frac{\partial \left(\sum_{k'=1}^{N^{(l)}} \sigma\left(a_{k'}^{(l)}\right) w_{k,k'}^{(l+1)} \right)}{\partial a_i^{(l)}} = \sigma'\left(a_i^{(l)}\right) w_{k,i}^{(l+1)} \tag{2.46}$$

and since $\sigma\left(a_{k'}^{(l)}\right)$ is independant of k the final recursive expression is found to be

$$\delta_i^{(l)} = \sigma'\left(a_i^{(l)}\right) \sum_{k=1}^{N^{(l+1)}} \delta_k^{(l+1)} w_{k,i}^{(l+1)} \quad \text{for} \quad l = 1, \ldots, L-1, \tag{2.47}$$

where the recursive dependency is shown explicitly, i.e. the errors $\delta_i^{(l)}$ depends on the errors from the later layer $\delta_k^{(l+1)}$ [5].

Finally having computed the errors of each neuron in the network, the gradients can be computed for each layer like for the output layer shown in (2.40)

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial w_{i,j}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)} \quad \text{for} \quad l = 1, \dots, L-1 \tag{2.48}$$

and hence making the training of the FFNN efficient.

2.2.2 Recurrent neural network

The FFNN presented in Section 2.2.1 is a very powerful model, but it has its limitations. One being that it is a memoryless model, i.e. for each input the network will have no information about previous inputs. This property however can be obtained using a "*Recurrent Neural Network*" (RNN).

RNN's are basically identical to FFNN's with the minor adjustement, that each layer l will not only pass its output to the next layer $l+1$, but will also pass it to itself, hence a "*recurrent connection*". This is visualized in Figure 2.5.

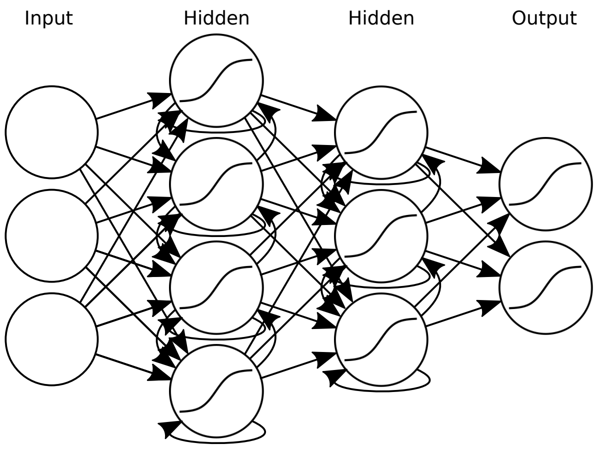


Figure 2.5: Visualization of a 3-layer recurrent neural network with input $\mathbf{x} \in \mathbb{R}^3$ and output $\mathbf{y} \in \mathbb{R}^2$.

A popular way of visualizing RNN's is by so-called "*unfolding*" the network and thereby explicitly show the recurrent connections between each time-step t. An example of the network shown in Figure 2.5 unfolded can be seen in Figure 2.6.

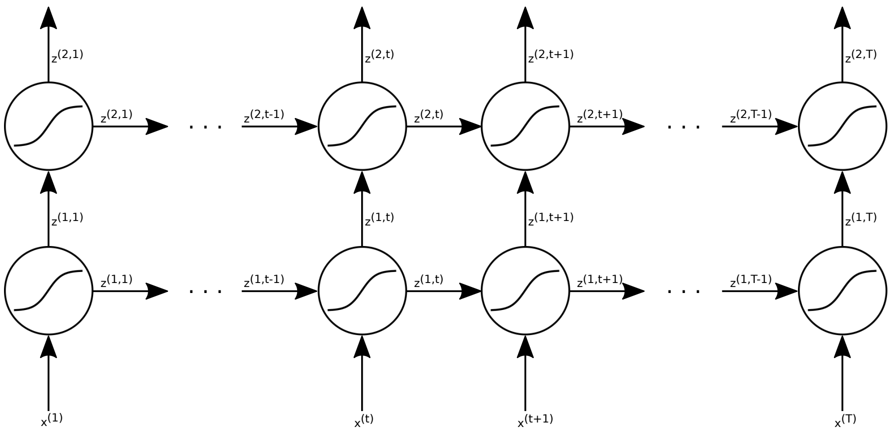


Figure 2.6: Visualization of an unfolded 3-layer recurrent neural network (2 recurrent layers and 1 output layer) with input sequences of length T. $\mathbf{x}^{(1)}$ represents the initial input at time-step $t = 1$ of a given observation $\mathbf{x}$ and $\mathbf{z}^{(2,T)}$ the final output at time-step $t = T$.

Let input variables be sequences of length T i.e. $\mathbf{x} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(T)}\}$ e.g. representing each character in a sentence of length T, and denote

$$\mathcal{U} = \{\mathrm{U}^{(1)}, \mathrm{U}^{(2)}, \dots, \mathrm{U}^{(L-1)}\} \tag{2.49}$$

the set of weights for the recurrent connections (the output layer has no recurrent connections) with $\mathrm{U}^{(l)} = \begin{bmatrix} \mathbf{u}_1^{(l)} & \mathbf{u}_2^{(l)} & \dots & \mathbf{u}_{N^{(l)}}^{(l)} \end{bmatrix}$ and $\mathbf{u}_k^{(l)} \in \mathbb{R}^{N^{(l)}}$. Then denote the activation of neuron k in layer l at a given time-step t expressed by extending (2.21) with a term for the recurrent connection

$$z_k^{(l,t)} = \sigma\left(a_k^{(l,t)}\right) = \sigma\left(\sum_{k'=1}^{N^{(l-1)}} z_{k'}^{(l-1,t)} w_{k,k'}^{(l)} + \underbrace{\sum_{k'=1}^{N^{(l)}} z_{k'}^{(l,t-1)} u_{k,k'}^{(l)}}_{\text{recurrent term}}\right) \tag{2.50}$$

which can be seen as introducing some kind of "*memory state*" to each layer, since it in theory will be able to remember all previous seen inputs [8].

Training RNN's can be done using a modified version of the backpropagation algorithm explained in Section 2.2.1.2, the so-called "*BackPropagation Through Time*" (BPTT) which involves computing modified versions of the error terms described in Section 2.2.1.2. The error terms are modified to their corresponding time-dependent versions $\delta_i^{(l,t)}$ by again adding a recurrent term to their definitions from (2.47)

$$\delta_k^{(l,t)} = \sigma'\left(a_k^{(l,t)}\right)\left(\sum_{k'=1}^{N^{(l+1)}} \delta_{k'}^{(l+1,t)} w_{k',k}^{(l+1)} + \underbrace{\sum_{k'=1}^{N^{(l)}} \delta_{k'}^{(l,t+1)} u_{k',k}^{(l)}}_{\text{recurrent term}}\right) \tag{2.51}$$

with $\delta_k^{(l,T+1)} = 0 \ \forall k, l$ for a complete definition of the recurrency. Finally the BPTT algorithm defines the gradients as summing the error terms for each time-step t [8]

$$\frac{\partial \mathcal{L}(\mathbf{x}, \mathbf{y})}{\partial w_{k,j}^{(l)}} = \sum_{t=1}^{T} \delta_k^{(l,t)} z_k^{(l,t)}. \tag{2.52}$$

Even though this recurrent time-dependency of RNN's make them capable of learning long-term dependency patterns, and even capable of utilizing the information of any input value it has ever seen in theory, this is found to not work be true in practice. Some of the problems and how to handle them will be discussed further in Section 2.2.4.

2.2.3 Bidirectional recurrent neural network

One restriction with a regular RNN is that at a given time-step t only information from previous time-steps are available to the network. "*Bidirectional Recurrent Neural Network*"s (BRNN's) make it possible to have both previous and future information available to the network at time t. This is done by feeding the sequence both in regular and reversed order to the network, each with it's own recurrent hidden layer, which then both are passed throughout the network [29]. This of course requires that the future time-steps are available at all time-steps t, hence it would not work e.g. for a decoder predicting one character at a time, and using the previous character to predict the next.

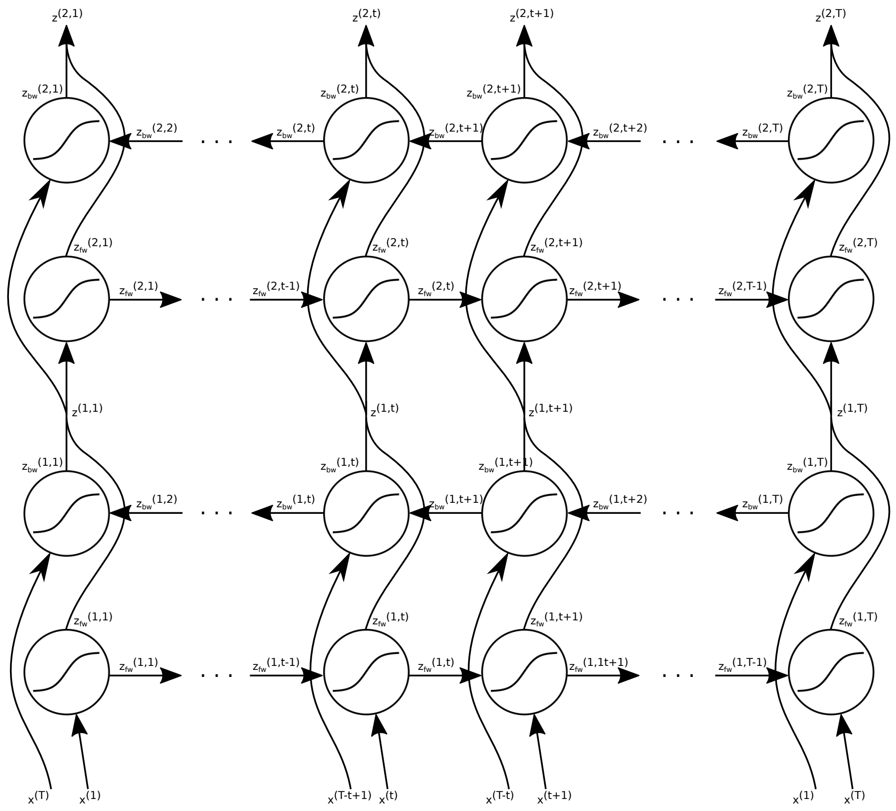


Figure 2.7: Visualization of an unfolded 3-layer bidirectional recurrent neural network with input sequences of length T. $z_{\text{fw}}^{(l,t)}$ and $z_{\text{bw}}^{(l,t)}$ denotes the forwards- and backwards passed activation of layer l at time t respectively. $z^{(l,t)}$ denotes the merged backards- and forwards passes.

A visualization of a BRNN can be seen in Figure 2.7 where it is shown that the output of the forwards- and backwards recurrent layers are concatenated before being fed as input to the next layer. This is one way of handling the extra hidden state from the backwards layer. Another way could be to project the merged outputs to the size of a single output layer, to maintain the state size throughout the network.

2.2.4 Long short-term memory

As mentioned in Section 2.2.2 the standard architecture of RNN's have certain problems with handling the recurrent information in the network. Specially learning long-term dependencies have been found to be a real struggle for regular RNN's in practice [24]. Also the fact that information is being fed back into it's own hidden

layer on each time-step has shown to either blow up or decay the outputs exponentially, which makes the training very hard. This is called the "*exploding gradient problem*" and the "*vanishing gradient problem*" [8].
To deal with these problems the very popluar so-called "*Long Short-Term Memory*" (LSTM) [13] architecture can be used. The LSTM both handles long-term dependencies, and preserves gradient information through time better than the regular RNN architecture [8], which have made it the preferred RNN architecture over regular RNN's.
Since the LSTM extends the regular RNN neuron to contain multiple operations, it makes sense to refer to each neuron in the RNN as a "*block*". A simple illustration of a regular RNN block can be seen in Figure 2.8.

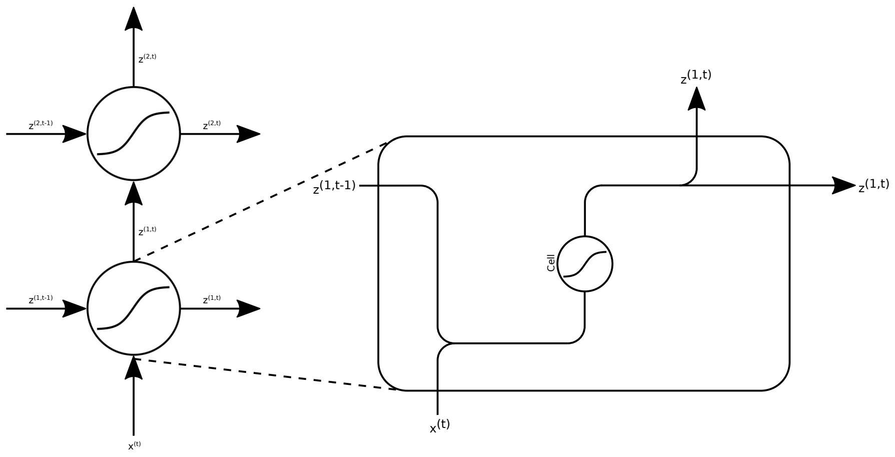


Figure 2.8: A regular RNN block with $\mathbf{x}^{(t)}$ and $\mathbf{z}^{(1,t)}$ being the input and output respectively, and $\mathbf{z}^{(1,t-1)}$ being the output from the previous time-step.

The LSTM extends the simple block shown in Figure 2.8 in multiple ways. Firstly a "*cell state*" $\mathbf{c}^{(t)} \in \mathbb{R}^C$, with C being the dimensionality of the state, is introduced which is capable of containing information through time if necessary. This can be seen as the "*memory*" of the LSTM block. Secondly a number of different "*gates*" are introduced which each takes the concatenated input $\mathbf{x}^{(t)}$ and previous output $\mathbf{z}^{(t-1)}$ as input and outputs a vector of size C with values in the range $]0, 1[$ using the sigmoid function and learned weights and biases W_f, W_i, W_o, and $\mathbf{b}_f$, $\mathbf{b}_i$, $\mathbf{b}_o$.
Letting g and h denote non-linear functions often chosen as tanh, and $\oplus$ and $\otimes$ denote element-wise addition and multiplication respectively, the gates and their purpose can be explained as:

- The **Forget gate**: Determines what information to forget by performing element-wise multiplication between the output of the gate $\mathbf{f}^{(t)}$ and the previous cell state $\mathbf{c}^{(t-1)}$.

- The **Input gate**: Determines what new information from the cell values $\tilde{\mathbf{c}}^{(t)} = g\left(\mathrm{W}_c\left[\mathbf{z}^{(t-1)}, \mathbf{x}^{(t)}\right] + \mathbf{b}_c\right)$ should be added to the cell state by performing element-wise addition between the updated cell state $\mathbf{c}^{(t-1)} \otimes \mathbf{f}^{(t)}$ and the modified cell values $\mathbf{i}^{(t)} \otimes \tilde{\mathbf{c}}^{(t)}$, resulting in the new cell state $\mathbf{c}^{(t)} = \mathbf{c}^{(t-1)} \otimes \mathbf{f}^{(t)} + \mathbf{i}^{(t)} \otimes \tilde{\mathbf{c}}^{(t)}$.

- The **Output gate**: Determines what values to output by performing element-wise multiplication between a non-linear transformation of the new cell state $h\left(\mathbf{c}^{(t)}\right)$ and the output values of the gate $\mathbf{o}^{(t)}$, leading to the final output of the block $\mathbf{z}^{(t)} = \mathbf{o}^{(t)} \otimes h\left(\mathbf{c}^{(t)}\right)$.

A visualization of the LSTM block can be seen in Figure 2.9 and the full set of equations is listed in (2.53) to (2.58).

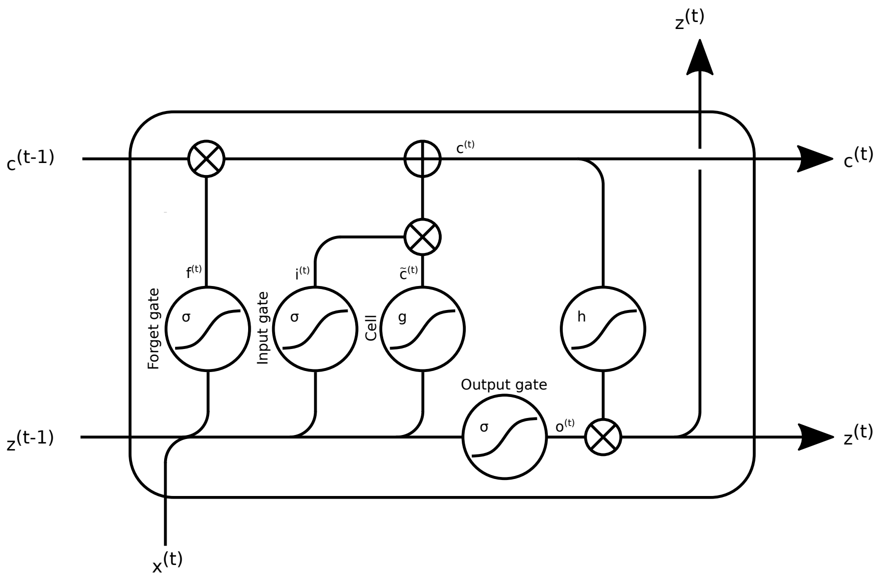


Figure 2.9: LSTM block with $\mathbf{x}^{(t)}$ and $\mathbf{z}^{(t)}$ being the input and output, and $\mathbf{c}^{(t)}$ being the cell state at time-step t. σ denotes sigmoid functions and g and h denotes non-linear functions. $\oplus$ denotes element-wise addition and $\otimes$ denotes element-wise multiplication.

Forget gate $$\mathbf{f}^{(t)} = \sigma\left(\mathrm{W}_f\left[\mathbf{z}^{(t-1)}, \mathbf{x}^{(t)}\right] + \mathbf{b}_f\right) \tag{2.53}$$

Input gate $$\mathbf{i}^{(t)} = \sigma\left(\mathrm{W}_i\left[\mathbf{z}^{(t-1)}, \mathbf{x}^{(t)}\right] + \mathbf{b}_i\right) \tag{2.54}$$

$$\tilde{\mathbf{c}}^{(t)} = g\left(\mathrm{W}_c\left[\mathbf{z}^{(t-1)}, \mathbf{x}^{(t)}\right] + \mathbf{b}_c\right) \tag{2.55}$$

State update $$\mathbf{c}^{(t)} = \mathbf{f}^{(t)} \otimes \mathbf{c}^{(t-1)} + \mathbf{i}^{(t)} \otimes \tilde{\mathbf{c}}^{(t)} \tag{2.56}$$

Output gate $$\mathbf{o}^{(t)} = \sigma\left(\mathrm{W}_o\left[\mathbf{z}^{(t-1)}, \mathbf{x}^{(t)}\right] + \mathbf{b}_o\right) \tag{2.57}$$

$$\mathbf{z}^{(t)} = \mathbf{o}^{(t)} \otimes h\left(\mathbf{c}^{(t)}\right). \tag{2.58}$$

For the function g and h in the input and output gate, the tanh function is chosen as activation function in this project.

2.3 Language Modelling

"*Language modelling*" is the task of learning the distribution of words or characters in a text corpus $\mathcal{D}$. Since this project concerns the language modelling on a character level, this section will use the same approach.
Let $\mathcal{A}$ be the set of unique characters in $\mathcal{D}$ refered to as the "*Alphabet*", and let $x^{(t)}$ denote a character in a sequence from $\mathcal{D}$, and let $c^{(t)}$ denote the context of $x^{(t)}$ (usually chosen as the previous n characters). In language modelling one then attempts to learn the probability distribution of the next character $x^{(t+1)}$ given the context $c^{(t)}$, i.e. one tries to learn the multinomial distribution $p\left(x^{(t+1)} \middle| c^{(t)}\right)$ [7].
During training $x^{(t)}$ is typically represented using a "*one-hot*" encoded vector of size $K = |\mathcal{A}|$, with each unique character acting as an independent class, hence allowing NN's to simply model $p\left(x^{(t+1)} \middle| c^{(t)}\right)$ like a regular classification task. E.g. chosing the context $c^{(t)}$ as the previous n characters a FFNN could parametrize $p\left(x^{(t+1)} \middle| c^{(t)}\right)$ using the context $x^{(1)}, \ldots, x^{(t)}$ as input to the network and using a softmax function in the output layer:

$$p\left(x^{(t+1)} = k \middle| x^{(t-n+1)}, \ldots, x^{(t)}\right) = \frac{\exp\left(z_k^{(L)}\right)}{\sum_{k'=1}^{K} \exp\left(z_{k'}^{(L)}\right)}. \tag{2.59}$$

The RNN however would not be restricted to using the previous n characters as the context, but would be able to use all previous characters (and future characters if using a BRNN), by feeding one character at a time to the network, and thereby having information available from all previous $t-1$ characters at time-step t, hence the parametrization would look as follows

$$p\left(x^{(t+1)} = k \middle| \, x^{(1)}, \ldots, x^{(t)}\right) = \frac{\exp\left(z_k^{(t+1,L)}\right)}{\sum_{k'=1}^{K} \exp\left(z_{k'}^{(t+1,L)}\right)}, \tag{2.60}$$

and for a BRNN

$$p\left(x^{(t+1)} = k \middle| \, x^{(1)}, \ldots, x^{(t)}, x^{(t+2)}, \ldots, x^{(T)}\right) = \frac{\exp\left(z_k^{(t+1,L)}\right)}{\sum_{k'=1}^{K} \exp\left(z_{k'}^{(t+1,L)}\right)} \tag{2.61}$$

for sequences of length T.

2.4 Sequence to sequence learning

Sequence to sequence learning (seq2seq) is the general task of mapping a given sequence to another sequence, which is the general purpose of a RNN. Instead of using a single RNN for the purpose, one can split the task into two steps as is presented in [32] using a different RNN for each task. The first RNN encodes a given sequence to a fixed length vector representation - hence refered to as the "*encoder*", and the second RNN decodes the vector to the target sequence. This is the so-called "*Encoder-Decoder*" framework.

An example of an input sequence ABC of length $T = 3$ encoded to a fixed vector $\mathbf{c}_{\text{ENC}}^{(T)}$ can be seen in Figure 2.10, where it is seen that the fixed vector $\mathbf{c}_{\text{ENC}}^{(T)}$ is taken as the state of the RNN at time-step T - the end of the sequence, which is represented to the network as an extra "*end-of-sequence*" class denoted the `<EOS>` token.

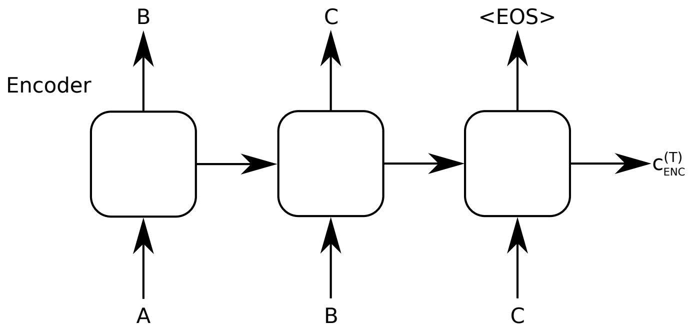


Figure 2.10: **Encoder**: Example of an encoder encoding the input sequence ABC to a fixed vector representation $\mathbf{c}_{\text{ENC}}^{(T)}$.

The state of the decoder is then initialized with the value of the encoded input sequence $\mathbf{c}_{\text{ENC}}^{(T)}$ and using an extra "*start-of-sequence*" class, denoted the `<GO>` token representing the beginning of the decoding, as the initial input decodes the target sequence. This is visualized in Figure 2.11

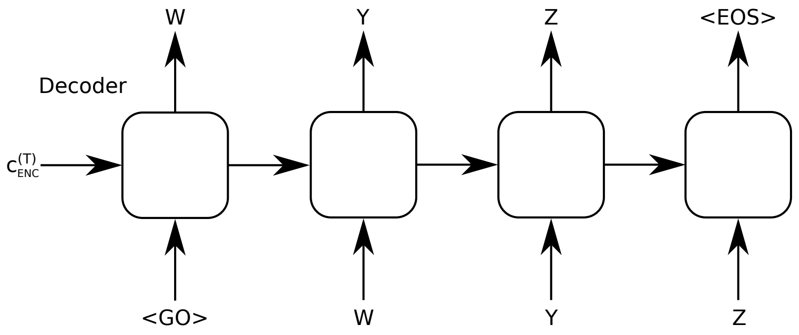


Figure 2.11: **Decoder**: Example of a decoder decoding the target sequence $WXYZ$ using the encoded fixed vector representation $\mathbf{c}_{\mathrm{ENC}}^{(T)}$ of the input sequence as initial hidden state.

2.5 Attention learning

The "*attention-mechanism*" presented in [2] has shown great performance gains in machine translation, and is now an often used method in the Encoder-Decoder framework. In a regular Encoder-Decoder setting the only information the decoder has from the encoded sequence is the encoded vector. Using attention the decoder gets information from the encoder at each time-step in the encoding process.
Given the hidden state vectors of the encoder at each encoding time-step

$$\left(\mathbf{c}_{\mathrm{ENC}}^{(1)}, \mathbf{c}_{\mathrm{ENC}}^{(2)}, \ldots, \mathbf{c}_{\mathrm{ENC}}^{(T_1)}\right), \qquad \text{with} \quad \mathbf{c}_{\mathrm{ENC}}^{(t)} \in \mathbb{R}^{N_1}\ \forall t \tag{2.62}$$

and hidden state vectors of the decoder at each decoding time-step

$$\left(\mathbf{c}_{\mathrm{DEC}}^{(1)}, \mathbf{c}_{\mathrm{DEC}}^{(2)}, \ldots, \mathbf{c}_{\mathrm{DEC}}^{(T_2)}\right), \qquad \text{with} \quad \mathbf{c}_{\mathrm{DEC}}^{(t)} \in \mathbb{R}^{N_2}\ \forall t, \tag{2.63}$$

learned parameters $\mathrm{W}_1 \in \mathbb{R}^{N_1 \times N_1}$, $\mathrm{W}_2 \in \mathbb{R}^{N_1 \times N_2}$, and $\mathbf{v} \in \mathbb{R}^{N_1}$ with N_1 and N_2 being the size of the hidden states of the encoder and decoder respectively.
An attention vector $\boldsymbol{\alpha}^{(t)}$ is computed for each time-step during decoding using

$$u_i^{(t)} = \mathbf{v}^T \tanh\left(\mathrm{W}_1 \mathbf{c}_{\mathrm{ENC}}^{(i)} + \mathrm{W}_2 \mathbf{c}_{\mathrm{DEC}}^{(t)}\right) \tag{2.64}$$

$$\alpha_i^{(t)} = \frac{\exp\left(u_i^{(t)}\right)}{\sum_{k=1}^{T_1} \exp\left(u_k^{(t)}\right)} \tag{2.65}$$

which is then used to compute a weighted sum of the encoder hidden states

$$\mathbf{d}^{(t)} = \sum_{i=1}^{T_1} \alpha_i^{(t)} \mathbf{c}_{\mathrm{ENC}}^{(i)} \tag{2.66}$$

which is concatenated to the output of the decoder at each decoding time-step, and thereby also used as input in future decoding time-steps.

2.6 Regularization

Regularization is a set of methods for restricting a models expressiveness in order to prevent it from overfitting. For neural networks the most common regularization methods are methods like

- **Weight decay**: Constraining the norm of the weight vectors/matrices using the L1 or L2 norm.
- **Dataset expansion**: Expanding the dataset e.g. by adding random noise to samples.
- **Constrained gradient norm**: Constraining the norm of the gradients.
- **Dropout**: Randomly disable neurons during training.
- **Early stopping**: Stop the network training when showing tendencies of overfitting.

In this project **Weight decay**, **Constrained gradient norm**, **Dropout**, and **Early stopping** have been used.

2.6.1 Weight decay

Weight decay, also referred to as "*L_2-regularization*", regularizes the network by constraining the L_2-norm (Euclidian norm) of a given set of weights, and thereby reduce extreme weight values thus reduce overfitting. The L_2-norm of a given weight vector $\mathbf{x} \in \mathbb{R}^p$ is given by

$$L_2\text{-norm} = \|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{p} x_j^2} = \sqrt{\mathbf{x}^T\mathbf{x}} \tag{2.67}$$

Given a weight matrix $\mathrm{X} \in \mathbb{R}^{p \times n}$ with columns $\{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ the L_2-norm is computed using the generalized L_2-norm namely the "*Frobenius norm*" given by

$$\text{Frobenius norm} = \|\mathrm{X}\|_\mathrm{F} = \sqrt{\sum_{i=1}^{n} \|\mathbf{x}_i\|_2^2} = \sqrt{\sum_{i=1}^{n} \mathbf{x}_i^T\mathbf{x}_i}. \tag{2.68}$$

In order to constrain the L_2-norm during learning the norm of the weights is added as an extra term to the loss function which is being minimized. I.e. given a loss function $\mathcal{L}$ the adjusted loss function constraining the L_2-norm of the weight matrix X is given as

$$\mathcal{L}_{\text{Regularized}} = \mathcal{L} + \lambda \left\| \mathrm{X} \right\|_{\mathrm{F}} \tag{2.69}$$

with λ being the "*regularization parameter*" controlling the amount of regularization. In order to reduce calculations and still obtain the same regularizing effect, the square root used in the calculation of the norm is often omitted by squaring the term, hence resulting in the following loss function

$$\mathcal{L}_{\text{Regularized}} = \mathcal{L} + \lambda \left\| \mathrm{X} \right\|_{\mathrm{F}}^2 \tag{2.70}$$

$$= \mathcal{L} + \lambda \sum_{i=1}^{n} \mathbf{x}_i^T \mathbf{x}_i. \tag{2.71}$$

2.6.2 Constrained gradient norm

By constraining the norm of the weight gradients in a network to a given threshold τ one limits the size of the "*step*" taken at each gradient descent step, hence preventing the model from taking any big steps in a wrong direction in the weight space.
Given a weight gradient $\nabla \mathrm{W}$ and a threshold τ the gradient is simply rescaled using

$$\nabla \mathrm{W} = \frac{\tau}{\left\| \nabla \mathrm{W} \right\|} \nabla \mathrm{W}. \tag{2.72}$$

Constraining the gradient norm can prevent both the vanishing gradients problem and the exploding gradients problem [26].

2.6.3 Dropout

Dropout was originally developed for FFNN's in [11] where it was shown, that randomly omitting half of the neurons in each layer during training significantly improved the performance of a network for different tasks.
Extending the dropout mechanism to RNN's was found to be non-trivial since applying dropout to RNN's naively showed poor performance for different tasks. In [36] it was shown how applying dropout to only the non-recurrent connections of a RNN improved the performance, since it didn't corrupt the "*memory*" of the network, hence this is the version of dropout used in this project.

2.6.4 Early stopping

Early stopping is a simple method for stopping the training of a network before the model starts overfitting, by evaluating and monitoring the training and validation error in each training epoch. The validation error is a small subset of the test dataset from which the validation error is computed. This subset is only used for determining the time to stop training which is when the validation error starts to increase. This is denoted the "*Sweet spot*" in Figure 2.12.

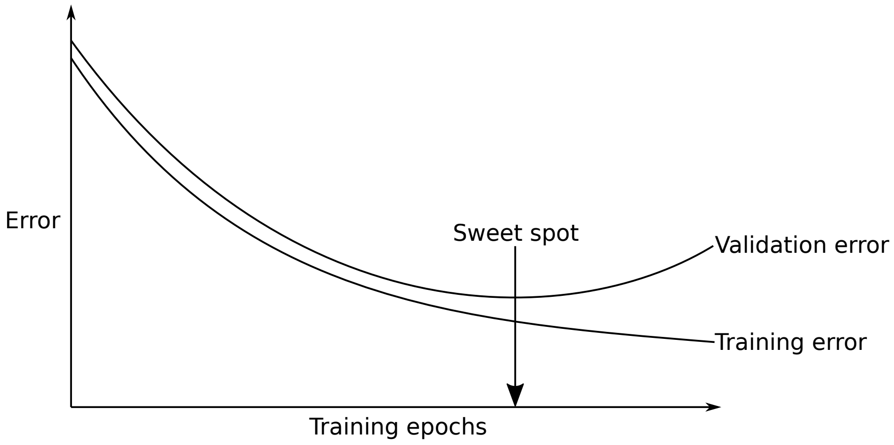


Figure 2.12: Early stopping during training based on the validation error. The "*Sweet spot*" indicates when the network starts to overfit the training data, hence is where the training is stopped.

This both prevents overfitting and reduces time of training at the cost of removing a small part of the dataset.

2.7 Batch normalization

A common pre-processing step when training neural networks is to normalize the training data to have 0 mean and a variance of 1. This makes it easier for the network to learn the appropriate weights, since it doesn't have to adjust its weights to fit the different offsets in the batches of the data.
However when training deep neural networks normalizing as a pre-processing step seems to be insufficient, since the offsets of the data change when propagating through the layers as the weights in the layers are changed.
In [16] a method for handling this issue is proposed, namely "*Batch normalization*". Instead of just normalizing the data before training, each mini-batch of training data is normalized before each layer in a network. I.e. given the input of a layer $\mathrm{X} \in \mathbb{R}^{n \times p}$ coming from a mini-batch of size n with each observation having p features, and let $\mathbf{x}_j$ be the j'th dimension of each observation in the mini-batch, then each dimension of the mini-batch is normalized using

$$\hat{\mathbf{x}}_j = \gamma_j \frac{\mathbf{x}_j - \mu_j}{\sigma_j} + \beta_j \tag{2.73}$$

with γ_j, μ_j, σ_j and β_j being learned parameters for each dimension with γ_j and σ_j being initialized at 1 and μ_j and β_j initialized at 0.
Batch normalization has shown to decrease training time of deep networks, increase their performance, and also increase training stability [16]. Furthermore it also acts as a regularizer.

2.8 Optimization

Training of neural networks consists of minimizing the networks objective function (loss function) w.r.t. the parameters of the network. This optimization is usually performed using gradient-based optimization algorithms such as "*Gradient descent*". For this project another gradient-based algorithm is used called the "*Adam*" algorithm.

2.8.1 Gradient descent

Given an objective function $L(\mathbf{w}) \in \mathbb{R}$ with $\mathbf{w} \in \mathbb{R}^p$ being the weights in a p-dimensional weight space, one can optimize $L(\mathbf{w})$ by iteratively moving in a direction determined from the direction of the gradient $\frac{\partial L(\mathbf{w}_i)}{\partial \mathbf{w}_i} = \nabla L(\mathbf{w}_i)$ at a given point $\mathbf{w}_i$ at iteration i with $\mathbf{w}_0$ being an initial guess of the weight values.
If one wants to minimize the objective function the update step at a given iteration i is given by

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \cdot \nabla L(\mathbf{w}_i). \tag{2.74}$$

with α being the step size often refered to as the "*learning rate*" in machine learning. The algorithm is described in Algorithm 1.

Algorithm 1 Gradient descent algorithm minimizing the loss function L using a convergence criteria of $\nabla L(\mathbf{w}_i) < \varepsilon$, and step size α.

```
 1: procedure GradientDescent($L$, $\alpha$, $\varepsilon$)
 2:     $\mathbf{w}_0 \leftarrow$ Initial guess (initialized at random)
 3:     $i \leftarrow 0$
 4:     while Not converged do
 5:         $\nabla L(\mathbf{w}_i) \leftarrow \frac{\partial L(\mathbf{w}_i)}{\partial \mathbf{w}_i}$
 6:         if $\|\nabla L(\mathbf{w}_i)\| < \varepsilon$ then
 7:             break
 8:         $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha \cdot \nabla L(\mathbf{w}_i)$
 9:         $i \leftarrow i + 1$
10:     return $\mathbf{w}_i$
```

A simple illustration of the application of the algorithm can be seen in Figure 2.13.

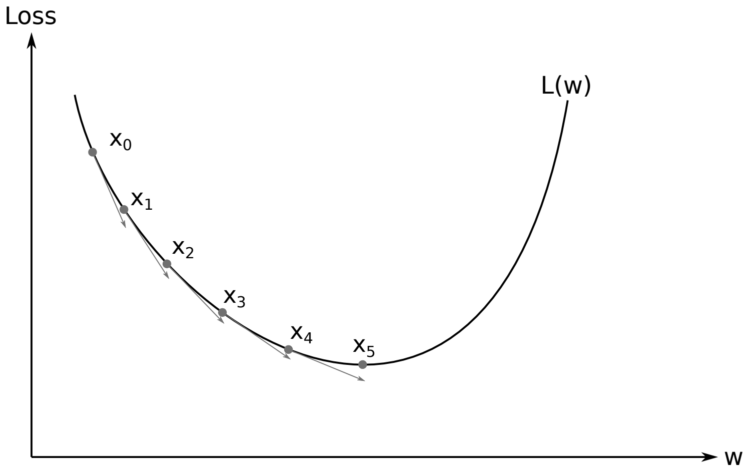


Figure 2.13: 1-dimensional example of gradient descent with x_0 being the initial value, $L(w)$ the loss function, and x_5 the final approximation to the minimum of $L(w)$. The small gray arrows indicate the step direction, i.e. the negative gradient.

2.8.2 The Adam algorithm

The Adam algorithm is a modification of the general gradient descent algorithm. It is is a combination of the two methods "*AdaGrad*" and "*RMSProp*" and is derived from Adaptive moment estimation, hence the name Adam [19]. It uses an approximation to the 1^{st} and 2^{nd} moment as moving averages of the gradient and of the squared gradient exponentially decaying w.r.t. the iteration number using decay rates β_1 and β_2 as hyper-parameters.

The algorithm is outlined in Algorithm 2.

Algorithm 2 The Adam algorithm minimizing the loss function L using a convergence criteria of $\nabla L(\mathbf{w}_i) < \varepsilon$, step size α, exponential decay rates $\beta_1, \beta_2 \in [0, 1)$ and an additional hyperparameter $\epsilon \ll 1$ [19].

```
1: procedure Adam($L$, $\alpha$, $\varepsilon$, $\beta_1$, $\beta_2$, $\epsilon$)
2:     $\mathbf{w}_0 \leftarrow$ Initial guess (initialized at random)
3:     $\mathbf{m}_0 \leftarrow \mathbf{0}$   (Initial 1st moment vector)
4:     $\mathbf{v}_0 \leftarrow \mathbf{0}$   (Initial 2nd moment vector)
5:     $i \leftarrow 0$
6:     while Not converged do
7:         $\nabla L(\mathbf{w}_i) \leftarrow \frac{\partial L(\mathbf{w}_i)}{\partial \mathbf{w}_i}$
8:         if $\|\nabla L(\mathbf{w}_i)\| < \varepsilon$ then
9:             break
10:        $\mathbf{m}_{i+1} \leftarrow \beta_1 \cdot \mathbf{m}_i + (1 - \beta_1) \cdot \nabla L(\mathbf{w}_i)$
11:        $\mathbf{v}_{i+1} \leftarrow \beta_2 \cdot \mathbf{v}_i + (1 - \beta_2) \cdot \nabla L(\mathbf{w}_i) \odot \nabla L(\mathbf{w}_i)$
12:        $\hat{\mathbf{m}}_{i+1} \leftarrow \frac{\mathbf{m}_{i+1}}{1-\beta_1^{i+1}}$
13:        $\hat{\mathbf{v}}_{i+1} \leftarrow \frac{\mathbf{v}_{i+1}}{1-\beta_2^{i+1}}$
14:        $\mathbf{w}_{i+1} \leftarrow \mathbf{w}_i - \alpha \cdot \frac{\hat{\mathbf{m}}_{i+1}}{\sqrt{\hat{\mathbf{v}}_{i+1}}+\epsilon}$
15:        $i \leftarrow i + 1$
16:    return $\mathbf{w}_i$
```

2.9 Bilingual evaluation understudy

The BiLingual Evaluation Understudy (BLEU) score proposed in [25] is a popular metric within machine translation for evaluating the similarity between two strings, hence is used to evaluate the quality of a generated translation vs. a true translation in NMT, based on n-gram precisions between the two. In order to penalize trivial candidates containing e.g. only the same word, which will get a high n-gram precision, a "*modified n-gram precision*" is used. The modified n-gram precision is defined as follows:

Given a candiate sentence C and a set of reference sentences $\mathcal{R} = \{R_1, R_2, \dots\}$ then for each n-gram in C count the maximum occurency in each reference R in $\mathcal{R}$ and the number of occurrences in the candidate C. These two counts are summed up for all n-grams in C and the modified precision is then given as the total number of maximum relevant n-gram occurrences in the references $\mathcal{R}$ divided by the total number of n-gram occurrences in the candidate.

The details are outlined in Algorithm 3.

Algorithm 3 The modified n-gram precision algorithm given a candidate sentence C and a set of reference sentences $\mathcal{R}$ [25].

```
procedure ModifiedNGramPrecision($C$, $\mathcal{R}$)
    $N \leftarrow 0$
    $M \leftarrow 0$
    for each unique $n$-gram $x \in C$ do
        $n \leftarrow 0$
        $m \leftarrow$ Number of occurrences of $x$ in $C$
        for each $R \in \mathcal{R}$ do
            $\hat{n} \leftarrow$ Number of occurrences of $x$ in $R$
            $n \leftarrow \max(n, \hat{n})$
        $N \leftarrow N + n$
        $M \leftarrow M + m$
    if $M > 0$ then
        $p \leftarrow \frac{N}{M}$
    else
        $p \leftarrow 0$
    return $p$
```

In order to take into account that short candidates can get a high precision even though the references are much longer, a penalty term is used in the calculations of the BLEU score refered to as the "*brevity penalty*".
Let c be the length of a candidate, and let r be the closest length to c of the reference lengths. Then the brevity penalty BP is defined as

$$\mathrm{BP} = \begin{cases} 1 & \text{if} \quad c > r \\ \exp\left(1 - \frac{r}{c}\right) & \text{if} \quad c \leq r. \end{cases}$$

The BLEU score is then calculated in the following way:
Let N be the user-defined maximum n-gram sizes used, and let p_n be the modified n-gram precision for a given candidate. The geometric mean is first computed for the n-gram precisions p_n

$$\left(\prod_{n=1}^{N} p_n\right)^{\frac{1}{N}} = \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log p_n\right) \tag{2.75}$$

which is changed to a weighted geometric mean by introducing weights w_n

$$\mathrm{GM} = \exp\left(\sum_{n=1}^{N} w_n \log p_n\right). \tag{2.76}$$

The final BLEU score is then computed as

$$\mathrm{BLEU} = \mathrm{BP} \cdot \mathrm{GM} \tag{2.77}$$

2.10 Levenshtein distance

The Levenshtein distance, usually just referred to as the "*Edit distance*", developed in [15] is used to compare two strings by calculating the number of required edits on one string to form the other. The allowed type of edits are:

- Insertion of a character.
- Deletion of a character.
- Substitution of a character with another.

Each edit contributes with a penalty of 1 to the total distance. E.g. the Levenshtein distance between "*eat*" and "*speak*" is 3 because it requires the 3 edits shown in Table 2.1:

#	**Edit**	**Character**	**Result**
1	Insert	"*p*"	"*peat*"
2	Insert	"*s*"	"*speat*"
3	Substitute	"*t*" with "*k*"	"*speak*"

Table 2.1: Example of required edit steps to compute the Levenshtein distance of 3 between "*eat*" and "*speak*".

2.11 Decision trees

Decision trees are rule-based decision making models which can be used for both regression and classification [10]. Considering classification trees they seek to split a set of N observations $\mathbf{x}_i$ each with p features and an associated class y_i

$$\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}, \qquad \mathbf{x}_i \in \mathbb{R}^p \tag{2.78}$$

by constructing a set of rules each splitting the observations based on the value of one the p features, resulting in a tree-like structure with each rule-split constructing two branches, and with the end-points (branches with no more splits) being the tree leaves (see Figure 2.14) from which each leaf determines a classification of the given input.

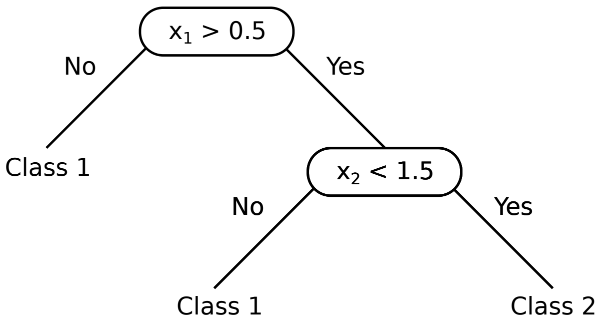


Figure 2.14: Example of a shallow decision tree used on a binary classification task.

For a K class problem an observation ending in a node (leaf) m is classified as the class belonging to the majority of observations from the dataset in the given leaf.
Let $\hat{p}_{m,k}$ denote the fraction of observations with class k in node m given as

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} I(y_i = k) \tag{2.79}$$

with R_m denoting the set of observations in node m, N_m the number of observations in node m, and I being an identity function being 1 if the statement is true and 0 otherwise.
Then an observation landing in node m is classified as

$$\text{class}\,(x \in R_m) = \underset{k}{\text{argmax}} \quad \hat{p}_{m,k} \tag{2.80}$$

In order to construct the rules making up the tree, a measure of the quality of a split is required. For classification trees the "*Gini impurity*" is usually used, which is given as

$$I_{\text{Gini}}(R_m) = \sum_{k=1}^{K} \hat{p}_{m,k}\,(1 - \hat{p}_{m,k}) \tag{2.81}$$

determining the "*impurity*" of a node m. For a given split splitting a set of observations into two sets R_i and R_j the impurity of the split is then quantified by the sum of their respective impurity i.e. $I_{\text{Gini}}(R_i) + I_{\text{Gini}}(R_j)$. Creating the best split-rule then comes down to choosing the split that results in the lowest impurity.
By computing the impurity for all feature values existing in the data an optimal split can be found from where the procedure can be repeated for each of the new nodes, and thereby iteratively building the tree. When a node contains less than a given number of observations the branch is considered saturated. This is a greedy algorithm but works very well in practice.
A popular strategy to learning decision trees is, to start by building a big tree (i.e. allow for many branches), and then prune the tree afterwards by collapsing internal nodes, that doesn't improve the misclassification rate enough [10].

2.12 Random forest

A Random forest is an ensemble method which consists of an ensemble of decision trees, i.e. it uses the "*Bootstrap aggregating*" (Bagging) concept which is the concept of averaging many unbiased and noisy models, and thereby reducing the overall variance. Since decision trees are approximately unbiased models, they are great candidates for the Bagging concept.
The training of each tree in a Random forest differes from training a regular decision tree in the following way:
Given a number of Random forest trees B each tree T_b is trained on a bootstrapped subset of the training data (sampling with replacement). For each creation of a best-split for tree T_b only a subset of the p features are taken into consideration, and the tree is only trained until a minimum number of nodes n_{min} is reached, i.e. only shallow trees are trained.
In order to classify a new observation using the Random forest, a simple majority vote is used, i.e. the class that most trees T_b predicts is chosen as the final prediction.

2.13 Transfer and Multi-task learning

2.13.1 Transfer learning

Transfer learning is the concept of making use of the knowledge gained from solving one task A to help one solve another task B. An example could be a neural network trained to recognize objects in images. This network could be useful in another context where one tries to locate dogs in an image for an example.
The use of pretrained ImageNet's is a great example of Transfer learning, since the weights learned have shown to generate high quality features that generalize very well to many types of images and tasks.
For NLP it is natural to expect solutions to tasks e.g. like spelling could be useful in solving other NLP tasks.

2.13.2 Multi-task learning

Multi-task learning is the concept of having a model learn multiple tasks simultaneously. The intuition behind the idea is, that some tasks might require the same basic understanding of the data on a lower level, e.g. two separate networks would learn the somewhat same weights in the lower layers, hence this part might be improved by sharing the weights between the two tasks.
Some research such as [9, 23] have already shown great results within Multi-task learning with using very few training examples. Solving multiple related tasks simultaneously seems to reduce the required amount of data.

CHAPTER 3

Methods

3.1 Implementation

Section 3.1.1 and Section 3.1.2 lists the software and hardware used for the project.

3.1.1 Software

The proposed and tested models have all been implemented using the `Tensorflow` library [22] for `Python` which allows for transparent use of highly optimized mathematical operations on GPU's through `Python`. A computational graph is defined in the Python script which defines all operations which are necessary for the specific computations. One then specifies the *tensors* wanted evaluated and `Tensorflow` runs the necessary part of the computational graph implemented in efficient `C` code on the CPU, or on a GPU if any is made availabe to the script and the operations have a supported GPU-implementation. A short example of the concept of using `Tensorflow` can be seen in Appendix B.2.

`Tensorflow` also supports the use of multiple GPU's however this was not utilized in this project, i.e. only a single GPU was used for any test run.

The plots for the report were generated using the `matplotlib` library for `Python`, and the illustrations have been created using `Inkscape`[1] which is a vector graphics software similar to Adobe Photoshop.

3.1.2 Hardware

The experiments were run on a machine with 12x "*Intel(R) Xeon(R)*" CPU's with specification listed in Table 3.1

[1] Inkscape is a free and open-source vector graphics editor. `https://inkscape.org/en/`

Specification	Value
Cores	6
Threads	12
Clock speed	3.50 GHz
Cache	15 MB SmartCache
Memory bandwidth	68 GB/s

Table 3.1: Specifications of the Intel(R) Xeon(R) CPU E5-1650 v3 from `https://ark.intel.com/products/82765/Intel-Xeon-Processor-E5-1650-v3-15M-Cache-3_50-GHz`

and on a single GPU of type "*Geforce GTX Titan X*" with specifications listed in Table 3.2

Specification	Value
CUDA cores	3072
Base clock speed	1000 MHz
Memory	12 GB
Memory clock speed	7.0 Gbps
Memory bandwidth	336.5 GB/s

Table 3.2: Specifications of the Geforce GTX Titan X from `http://www.geforce.com/hardware/desktop-gpus/geforce-gtx-titan-x/specifications`

The use of a GPU decreased the training time of the models with approximately a factor of 3, however this speed-up was not closely monitored throughout the project, hence it could have varied.

3.2 Data

For this project the following four text corpus' have been used:

- **The Q/A dataset**: Question/answer pairs from medical/health-related questions.
- **The extended Q/A dataset**: An extension of the Q/A dataset.
- **The PubMed dataset**: Text from medical papers and medical/health-related Wikipedia articles.
- **The Europarl English-French dataset**: English texts from proceedings from the European Parliament and their corresponding French translations.

The datasets are described in further detail in Sections 3.2.1 to 3.2.4

3.2.1 The Q/A dataset

The question/answer (Q/A) pairs are scraped from WebMD Answers[2], a website containing categorised questions asked by the users and their associated answers given by experts. The following is an example of a Q/A pair:

Q: "*I have a wart on my cheek it has been there for only two weeks. Is there a homeopathic cure I can try?*"

A: "*Hi you may find you answer on `www.earthclinic.com`, if not it would be a good starting place. Good luck*"

Each question can have multiple answers from specialists or simply from other users, and each answer therefore make up a question/answer pair with the given question.

In an attempt to only obtain good quality answers the answers coming from a user and not a specialist are filtered based on the users number of posts, votes, or followers which is available through the page.

An answer from a user is considered valid if one of the following criterias are meet:

- The user have written more than 50 answers.
- The user have more than 10 votes.
- The user have more than 5 followers.
- Other users have marked the answer as useful.

These requirements were made up based on intuition from browsing the forum, and thereby doesn't guarantee any quality improvement.

The length of the question and answers vary from pair to pair, and for some answers it varies a lot, hence for training the answers are truncated to only contain the first 3 sentences of the answer. The length distributions can be seen in Figure 3.1.

[2] `http://answers.webmd.com`

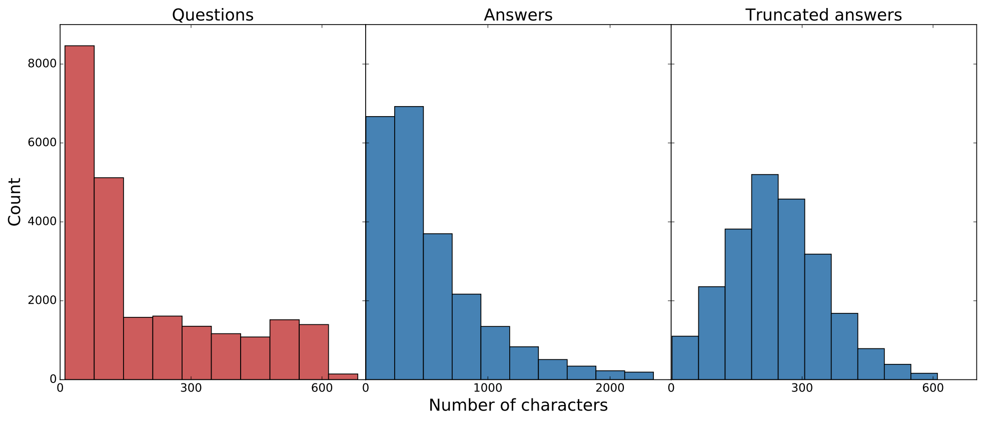


Figure 3.1: Sequence length distributions of the questions and answers in the Q/A dataset. The truncated answers are answers limitted to the first 3 sentences.

After removal of duplicated questions the dataset contained 23.437 Q/A pairs and a total of approximately 18 million characters.[3] From the final text corpus 17.843 Q/A pairs have an associated category from 77 different categories (listed in Appendix B.1). The category of each Q/A was determined from the associated tags [4], by matching them with records from a "*Unified Medical Language System*" (UMLS) database. The UMLS database contains records of symptoms and diseases grouped in a set of overall categories, which are the ones used as categories. From a given tag x the symptom/disease with the most similar name are chosen as the category.
To determine the similarity between tags and symptom/disease names the "*Levenshtein distance*", explained in Section 2.10, is used.
The distribution of assigned categories can be seen in Appendix B.1.1.

3.2.2 The extended Q/A dataset

In an attempt to extend the Q/A dataset the following additional websites were scraped for Q/A pairs[5]:

- eHealth forum[6]
- HealthTap[7]
- iCliniq[8]

[3] Final scraping done the 5th of May 2017.
[4] Users can add tags to questions when posting them on the Webmd forum.
[5] Data publicly available at `https://github.com/LasseRegin/medical-question-answer-data`
[6] `http://ehealthforum.com/health/health_forums.html`
[7] `https://www.healthtap.com/answers_by_specialty`
[8] `https://www.icliniq.com/qa`

- Question Doctors[9]

which resulted in a total of 166.804 Q/A pairs with a total of 66 million characters and thereby an increase in number of observations of approximately 700%, however also increasing the diversity of the data, since they origin from different websites. The length distributions of the extended dataset can be seen in Figure 3.2.

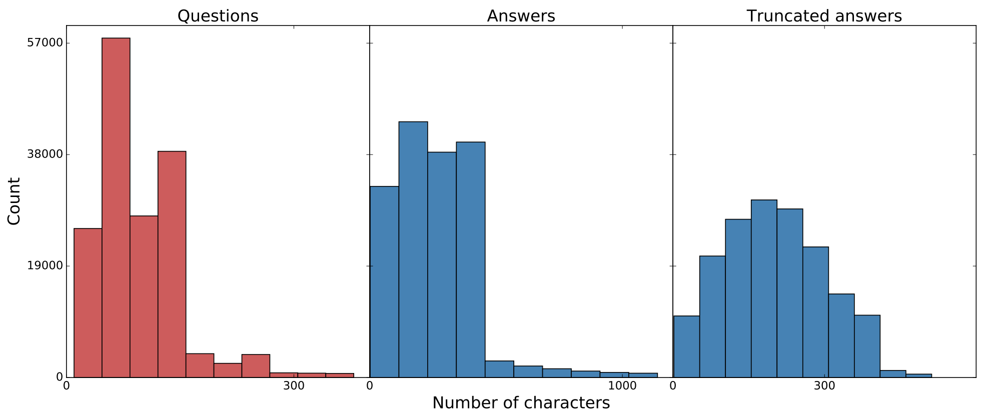


Figure 3.2: Sequence length distributions of the questions and answers in the extended Q/A dataset. The truncated answers are answers limitted to the first 3 sentences.

In order to compare the question and answer lengths of the newly introduced sources, the distribution of questions for each of them is plotted in Figure 3.3.

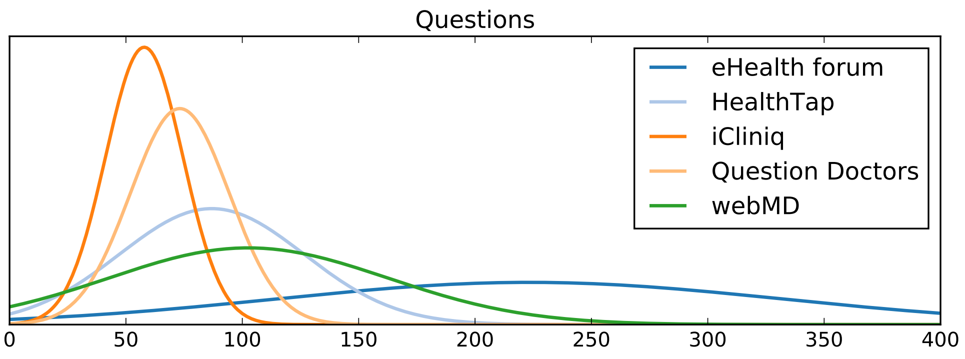


Figure 3.3: Normal distributions fitted to the sequence lengths of the questions for each of the new sources.

[9] `https://questiondoctors.com/blog/`

and the distribution of truncated answers for each of them is plotted in Figure 3.4

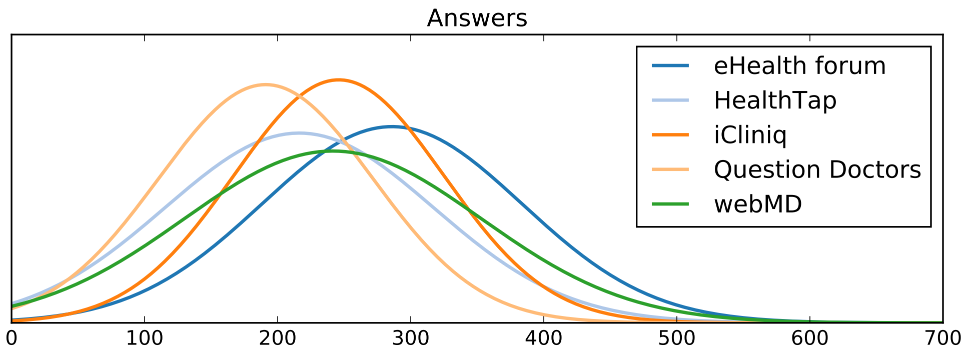


Figure 3.4: Normal distributions fitted to the sequence lengths of the truncated answers for each of the new sources.

where it is seen, that the mean question length from the eHealth forum source is quite longer than the other sources. The answer lengths doesn't differ too much.

3.2.3 The PubMed dataset

The PubMed dataset contains a mix of text from papers with medicine/health-related topics, and text from Wikipedia articles.

The papers are gathered from PubMed[10] and both their abstract and body text is used from a total of 53.541 papers.

The Wikipedia articles mentioned are all articles within the parent category "*Health and fitness*"[11] and within all of its subcategories, which resulted in a total of 7.077 articles.

A set of example observations can be seen in Table 3.3.

[10] `https://www.ncbi.nlm.nih.gov/pubmed/`
[11] `https://en.wikipedia.org/wiki/Portal:Contents/Health_and_fitness`

Example 1	Introduction Renal cell carcinoma (RCC) accounts for 3% of all adult malignancies and $25-30\%$ of RCC patients have metastatic disease at presentation.
Example 2	Nonsurgical treatment like laser ablation treatment (LAT) has also been used in toxic adenoma: compared with 131I, LAT obtained a similar volume reduction but seemed inferior in normalization of serum TSH [5].
Example 3	2-1-1 is a special abbreviated telephone number reserved in canada and the united states as an easy-to-remember three-digit telephone number meant to provide information and referrals to health, human and social service organizations.

Table 3.3: Examples of observations from the PubMed dataset.

The papers and Wikipedia articles results in a total of approximately 600 million characters.

3.2.4 The Europarl English-French dataset

The Europarl English-French dataset was created in [20] with the purpose of creating a benchmarking dataset for Statistical Machine Translation. The dataset contains proceedings from the European Parliament in 21 different European languages and with their corresponding english translations.
In this project the French translations are used in order to verify that the networks performed properly on a known dataset, with known state-of-the-art performance values.
A few translation pair examples can be seen in Tables 3.4 to 3.6

English:	My question relates to something that will come up on Thursday and which I will then raise again.
French:	Ma question porte sur un sujet qui est à l'ordre du jour du jeudi et que je soulèverai donc une nouvelle fois.

Table 3.4: Example of English-French translation pairs from the Europarl English-French dataset.

English:	Why have the staircases not been improved since my accident?
French:	Comment se fait-il que les escaliers n'aient pas été améliorés depuis mon accident?

Table 3.5: Example of English-French translation pairs from the Europarl English-French dataset.

English:	I admit that, at present, the matter seems to be somewhat confused.
French:	Je vous avouerai que, pour le moment, les choses me semblent un petit peu confuses.

Table 3.6: Example of English-French translation pairs from the Europarl English-French dataset.

The Europarl English-French dataset contains 2.007.723 translation pairs and a total of approximately 630 million characters.

3.2.5 Pre-processing

Since the data was gathered from online forums there was a lot of noise in the data, which was attempted removed. This included possible advertisements/spam which often could be filtered away by checking the fraction of uppercase letters in the post. Also questions containing only a link were removed.
The following pre-processing was performed on each question and answer in the Q/A dataset in order to clean up the data and remove noise:

- Convert to lowercase.
- Html/XML tags removed.
- Replace links with "*<link>*".
- Replace happy and sad emoticons with "*<positive_smiley>*" and "*<negative_smiley>*".
- Remove certain substrings from beginning of string which were manually determined from inspection of the gathered data. This mostly consisted of formal strings such as: "*I understand your pain...*", "*I understand what you are going through...*" etc. which occurred multiple times in the dataset.
- Replace multiple whitespaces with a single whitespace.
- Replace multiple symbols such as periods, exclamation marks, and question marks with a single symbol.
- Remove commas.
- Replace different types of apostrophes with a single type, to prevent multiple type of apostrophes in the text.

3.3 Models

This section presents the different model architectures tested in this project, and details about their implementation.
Each model will be using the general "*Encoder-Decoder*" setup discussed in Section 2.4 but each with a different architecture in either the encoder or decoder. The models will be operating on a character level by learning character embeddings, and thereby using a small "*alphabet*" of characters instead of a (usually very big) "*vocabulary*" of words which is often used in these kind of models. Batch normalization will be used on the character embeddings before feeding them in to the encoder.
Instead of including all unique characters from the dataset in the vocabulary a fixed alphabet was chosen prior to training, and characters not existing in the alphabet was given an "*unknown-character*" token. The fixed alphabet was chosen to be the following characters

```
abcdefghijklmnopqrstuvwxyz0123456789.,?:!-"()/
```

which covers 99.9% of all characters and resulted in an alphabet of size 53 when including the tokens for "*padding*", "*start-of-sequence*", "*end-of-sequence*", and "*unknown-character*".
The different models will be referred to as

- **LSTM**: Using regular LSTM cells in encoder and decoder.
- **BLSTM**: Using Bidirectional LSTM cells in the encoder.
- **BLSTM Attention**: Using Bidirectional LSTM cells in the encoder and attention in the decoder.
- **BLSTM Attention C2W**: Using a hierarchical char2word encoder and attention in the decoder.
- **Mixed LM**: Using decoder with a "*Language model mode*" and a "*generate answers mode*".
- **Assisted LM**: A model assisted by a pretrained language model.
- **Multi-task classifier**: A model classifying the question category and generating answers simultaneously.

3.3.1 LSTM

The LSTM model simply uses the LSTM cells explained in Section 2.2.4 in the encoder and decoder.
The architecture is visualized in Figure 3.5.

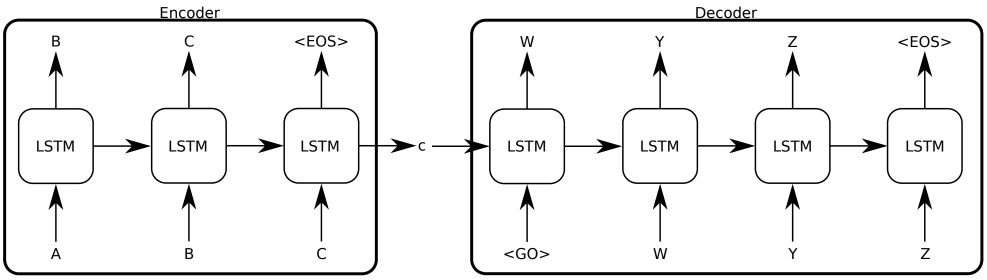


Figure 3.5: Graphical representation of the LSTM model.

3.3.2 BLSTM

The BLSTM model uses the Bidirectional RNN architecture explained in Section 2.2.3 with LSTM cells in the encoder, and for the decoder it uses regular LSTM cells.

The architecture is visualized in Figure 3.6.

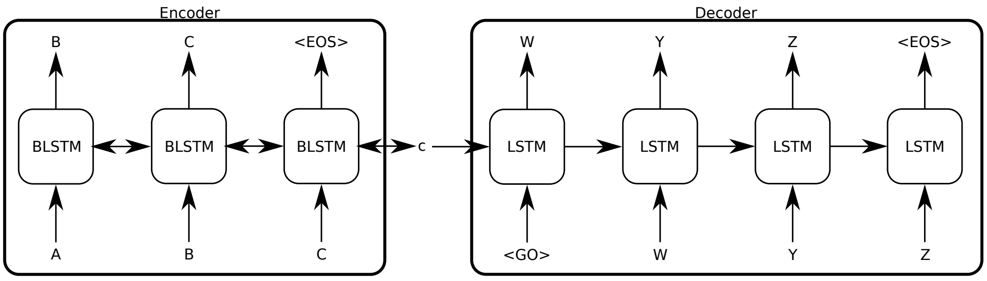


Figure 3.6: Graphical representation of the BLSTM model.

3.3.3 BLSTM Attention

The BLSTM Attention model uses the same encoder as the BLSTM model but uses the attention mechanism explained in Section 2.5 for the decoder.

The architecture is visualized in Figure 3.7.

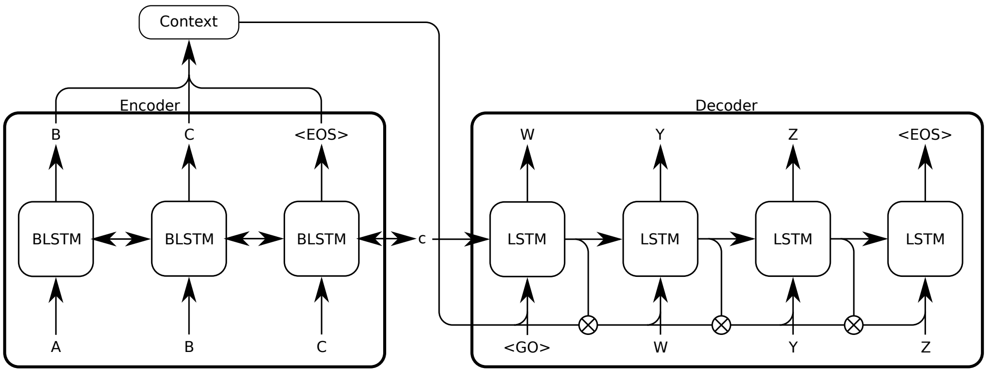


Figure 3.7: Graphical representation of the BLSTM Attention model.

3.3.4 BLSTM Attention C2W

The BLSTM Attention C2W (Char2Word) model uses a hierarchical char2word encoder proposed in [18] where an extra level of encoding is placed on top of the character-based RNN encoder. For each time-step containing an end of a word in the character-level encoder, the hidden state is feed in to the word-level encoder representing the encoding of the given word.
The architecture is visualized in Figure 3.8

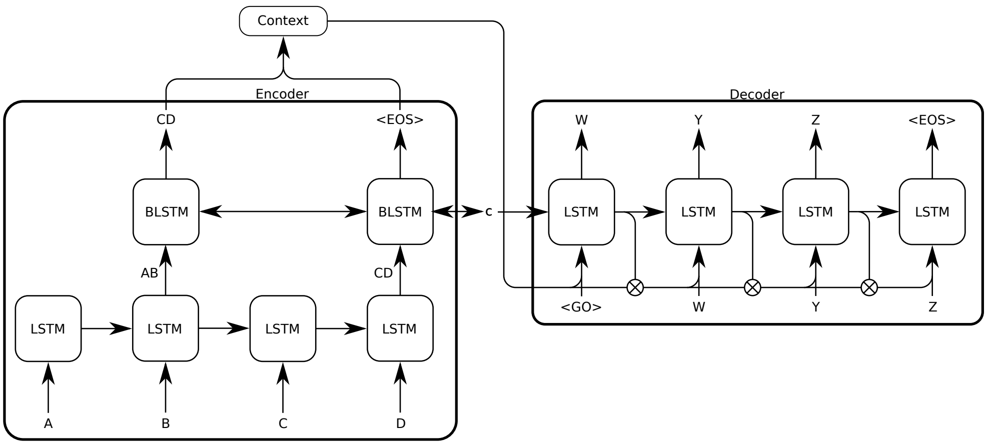


Figure 3.8: Graphical representation of the BLSTM Attention C2W model. "*AB*" and "*CD*" illustrates the encoded words.

Note that in the illustration there is no word-separating token (e.g. a space), however a word-separating token will be present in the real dataset.

3.3.5 Mixed LM

The Mixed LM model can be any of the previous models with an additional "*mode*" input for the decoder. The mode input is a binary variable representing what mode the decoder should run in. The two possible modes are "*Language model mode*" and "*Answer generation mode*", where the Answer generation mode will use the encoded question as initial state in the decoder, and the Language model mode will initialize the decoder state with zeros.
During training the batches will be a mix of observations from the Q/A dataset and observations from the PubMed dataset, with each observation having the binary mode parameter appended to it. The ratio between the two types will be denoted α_{mix}, i.e. α_{mix} denotes the fraction of Q/A observations used in each batch.
During training of a mixed LM model the validation loss used to determine convergence is only computed using Q/A observations, since this is the main task being solved.

3.3.6 Assisted LM

The Assisted LM model simply uses a pretrained language model to assist it during prediction. At each prediction time step the value is fed into the answer generation model and the language model, and the probability for each class predicted by the two models are merged by multiplying the probabilities, and finally renormalizing the probabilities, so they sum to 1.
Since this doesn't depend on the architecture of the network, the answer generation model can be any of the previous mentioned models.

3.3.7 Multi-task classifier

The Multi-task classifier attempts to classify the category of the question based on the final encoder state after encoding.
Given the final encoder state $\mathbf{c}^{(T)} \in \mathbb{R}^N$ and learned weight matrix and bias vector $\mathrm{W}_\mathrm{Q} \in \mathbb{R}^{Q \times N}, \mathbf{b}_\mathrm{Q} \in \mathbb{R}^Q$ of a dense layer with ReLU activation function taking $\mathbf{c}^{(T)}$ as input, with Q being a hyperparameter defining the number of units in the layer.
Furthermore let $\mathrm{W}_{\text{logits}} \in \mathbb{R}^{N_C \times Q}, \mathbf{b}_{\text{logits}} \in \mathbb{R}^{N_C}$ be a final projection of the encoder state to match the projection dimensionality with the number of question categories N_C, hence making up the logits of the category classes.
I.e. the logits used to predict the category classes are given from the following set of operations on the final encoder state $\mathbf{c}^{(T)}$

$$\mathbf{z}_Q = \max\left(0, \mathrm{W}_\mathrm{Q}\mathbf{c}^{(T)} + \mathbf{b}_\mathrm{Q}\right) \tag{3.1}$$

$$\text{logits} = \mathrm{W}_{\text{logits}}\mathbf{z}_Q + \mathbf{b}_{\text{logits}}. \tag{3.2}$$

The additional projections of $\mathbf{c}^{(T)}$ can be seen in Figure 3.9.

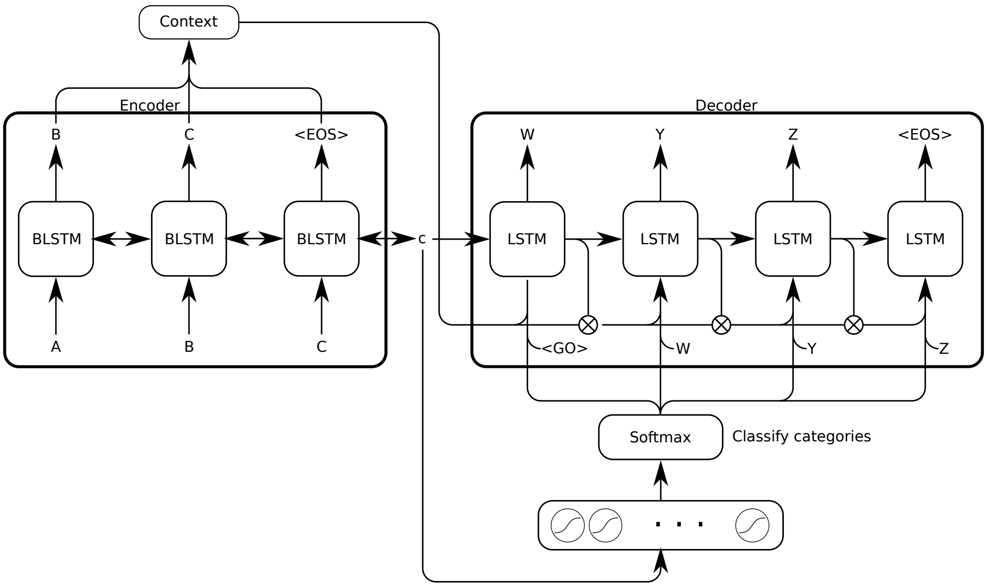


Figure 3.9: Graphical representation of the BLSTM Attention Classifier model.

From the logits the network is then able to learn to classify the category by extending the loss function

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, q) = -\sum_{k=1}^{K} y_k \ln z_k^{(L)} - \ln \text{logits}_q \tag{3.3}$$

with q denoting the question category index. In order to weight the importance of classifying vs. generating answers a hyperparameter $\alpha \in [0, 1]$ is introduced leading to the loss function

$$\mathcal{L}(\mathbf{x}, \mathbf{y}, q) = -\alpha \sum_{k=1}^{K} y_k \ln z_k^{(L)} - (1-\alpha) \ln \text{logits}_q. \tag{3.4}$$

In order to make the category information available to the decoder, the class probabilities, computed by applying a softmax function to the logits, are concatenated to the decoder inputs. During training the correct category class is used instead of the computed probabilities, encoded as a one-hot vector.
During training an observation is only allowed to have one associated category. Since some questions have multiple categories the category marked as the "*correct*" category is sampled from the given categories each time the observation is shown to the network. This will allow for the network to learn similarities between categories, and will act as a regularizer. This will however also make it harder for the network to obtain perfect accuracy, since a guess on A for an observation with category A and B might be considered incorrect during training if the sampled category is B.

For observations with no associated category the category probabilities are concatenated to the decoder input instead of the one-hot encoded vector, and the extra category prediction term in the loss function is set to 0 for the given observation.

3.4 Evaluation metrics

For comparing the performance of the different models the BLEU score explained in Section 2.9 was used.

3.5 Prediction

The prediction or inference phase refers to the generation of an answer candidate from a given question and is the main job of the decoder.
The standard procedure is to encode the question using the encoder, and set the initial state of the decoder to the final state of the encoder. By feeding the decoder with the `<GO>` token the decoding begins. At each time-step the decoder computes the probabilities of each token in the alphabet, from a given input token, and selects the token with highest probability as its prediction hence using it as input to the next time-step. This is repeated until the `<EOS>` is predicted.
This is a greedy approach since taking the one with highest probability at each time-step doesn't guarantee the most probable sequence to be chosen. Therefore two other prediction approaches are explored namely:

- **Beam search**
- **Sampling**

3.5.1 Beam search

Beam search is a graph search algorithm based on breadth-first search. Given a graph G, a number of partial candidates to maintain B and a initial node, the algorithm searches through one level of neighbour nodes of the initial node at a time keeping the B most optimal nodes and discarding the rest at each step. For each of the B kept nodes the neighbour nodes are then repeatedly searched again using the same approach until all nodes have been visited, leaving a set of B optimal candidates.
In [32] a so-called "*left-to-right beam search decoder*" is used, and shown to improve the BLEU score in translation by approximately 1%.
The left-to-right beam search decoder keeps the best B partial sequence candidates, refered to as the Beam, at each time-step, and extends them with all possible tokens from the alphabet. If the `<EOS>` token is predicted for one of the candidates the candidate is removed from the beam and marked as a finished candidate.

3.5.2 Sampling

The network decoder outputs a probability distribution over the tokens in the alphabet at each time-step, hence it is possible to draw a token from the given distribution, instead of just using the token associated with the highest probability.

By sampling B samples from a multinomial distribution given by the probabilities of the decoder at time-step 0, and then sample a single token for each sample for the remainding time-steps one obtains a sample of B diverse candidates.

However this approach has the risk of ending in a sequence of tokens the decoder have never seen during training, hence yielding poor sequence candidates.

An approach that might reduce this risk is to only sample in the early time-steps and change to using the most likely token at the last time-steps. An approach proposed in this project and refered to as "*Decaying sampling*" is explained in Section 3.5.2.1.

3.5.2.1 Decaying sampling

The Decaying sampling approach lets the sampling from the multinomial distribution, given by the class probabilities from the decoder, go from regular sampling towards the `argmax` function as the time-step increases. I.e. in early time-steps the sampling will be like drawing a sample from the distribution, and when the time-step gets close to the maximum number of time-steps T it will act as the `argmax` function, and use the class with highest probability.

Given class probabilities $\mathbf{p} \in [0,1]^K$ with K classes, the probabilities used for the multinomial distribution, from which the prediction is sampled from, denoted $\tilde{\mathbf{p}}$ at a given time-step t are given as

$$\tilde{p}_k = p_k^{1+\gamma \frac{t}{T}} \qquad \text{for} \quad k = 1 \dots K \tag{3.5}$$

$$\text{Renormalization} \qquad \tilde{p}_k = \frac{\tilde{p_k}}{\sum_{k'=1}^{K} p_{k'}} \qquad \text{for} \quad k = 1 \dots K \tag{3.6}$$

with γ being a decay factor and T the maximum allowed time-step.

A visualization of how the probabilities will converge towards just one of the classes having probability very close to 1 is shown in Figure 3.10.

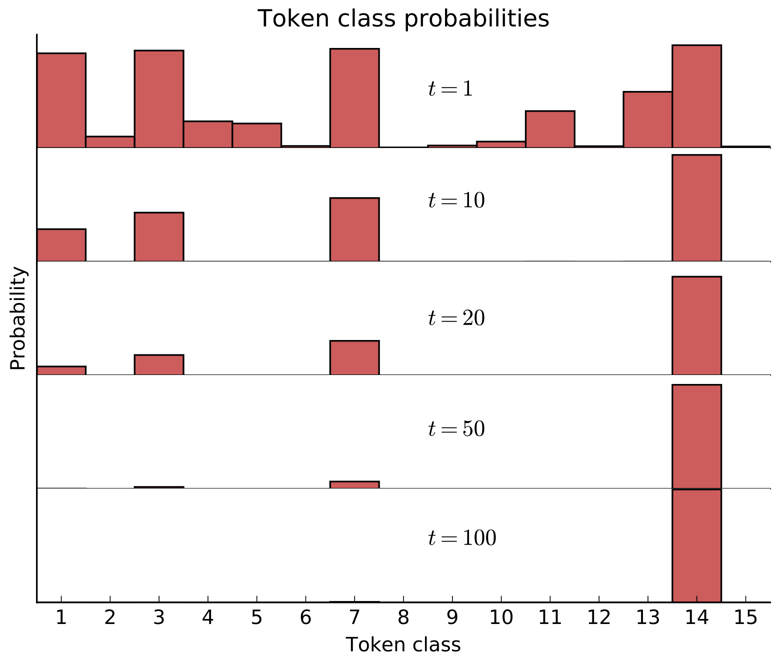


Figure 3.10: Example of a given set of class probabilities for 15 classes at different time-steps using the Decaying sampling approach with a decay factor of $\gamma = 150$ and a maximum number of time-steps of $T = 100$.

The motivation behind this is that during inference from a given encoded question the network might have learned a couple of possible answers that fit the given question, i.e. there might be a couple of token classes with high probability in the first time-steps which all will be ignored but the highest one when using `argmax`. By using sampling in the first time-steps instead some of these potential possible answers should be generated, and in order to not diverge completely from the "*known*" state space, the highest probable class is favoured increasingly with time.

3.5.2.2 Choosing optimal candidate

Using Beam Search or Decaying sampling B candidates are generated from which a optimal candidate must be found, since providing multiple answers is not a possibility. Determining the optimal candidate is done using two Random Forest classification models explained in Section 2.12.
One Random Forest RF_{question} is trained to predict which category a question belongs to, and another RF_{answer} is trained to predict which category the question which an answer is answering belongs to, i.e. RF_{question} is trained on questions and RF_{answer} is trained on answers.

Then given a question q with an associated answer a and B generated candidates c_b, RF_{answer} computes the category probabilities $\mathbf{p}_{\text{answer}}(c_b)$ for all candidates c_b, and RF_{question} computes the category probabilities for the question $\mathbf{p}_{\text{question}}$. The optimal candidate C is then determined as the candidate minimizing the euclidian distance between the two probability vectors

$$C = \underset{b}{\text{argmin}} \quad \left\| \mathbf{p}_{\text{answer}}(c_b) - \mathbf{p}_{\text{question}} \right\|_2 \tag{3.7}$$

3.6 Training

The training of the networks was performed using mini-batches of size 128 with the Adam algorithm explained in Section 2.8.2. Data was plit into training (70%), validation (10%), and test (20%), and the networks were all regularized using early stopping (Section 2.6.4) based on the validation set and with gradient norms constrainted to 5.0 (Section 2.6.2), and for some of the networks the dropout mechanism explained in Section 2.6.3, and weight decay explained in Section 2.6.1 was applied.
The network parameters were all inititalized at random from a uniform distribution in the interval $\left[-\frac{\sqrt{3}}{\sqrt{d}}, \frac{\sqrt{3}}{\sqrt{d}}\right]$ with d being the dimensionality of the input at a given layer which have been shown to be a good initialization strategy for deep networks [31].

3.6.1 Problem with Maximum Likelihood training

A general problem with training RNN's by feeding the true token to the network at each time-step is the discrepancy between training and inference/prediction. During training the network always gets the correct token as input at a given time-step t independently of what the predicted token in time-step $t-1$ was, hence it's mistakes does not propagate through time.
During inference the true token is not available hence the network always gets the previous predicted token as input. I.e. when the networks makes a mistake by predicting an incorrect token, the mistake propagates throughout the time-steps, and the predicted sequence diverges completely from the correct; Especially if the incorrect prediction is in an early time-step.
This problem is stated in [4] and an alternative training strategy claimed to solve the problem, refered to as "*Scheduled Sampling*", is proposed.

3.6.2 Scheduled Sampling

Scheduled Sampling is a training strategy for RNN's proposed in [4] which uses a mix between the correct token and the previously predicted token as input at a given time-step t during training.

Given the i^{th} mini-batch out of an estimated $i_{\max}$ number of mini-batches required for convergence then for each time-step t during training denote ε_i the probability of using the correct token as input with $1 - \varepsilon_i$ being the probability of using the infered token from previous time-step.
I.e. when $\varepsilon_i = 1$ the correct token is used a each time-step and when $\varepsilon_i = 0$ the previous predicted token is used.
By letting ε_i start with a value of 1 in the beginning of training and slowly let it converge towards 0 as the number of mini-batches trained i goes towards the estimated maximum number needed for converging $i_{\max}$, the training slowly becomes more similar to inference. This helps the network to learn correcting it's own mistakes [4].
The decaying schedules for ε_i proposed in [4] includes

- **Linear decay**: $\varepsilon_i = \max(\epsilon, b - ai)$ with $0 \leq \epsilon < 1$ being the minimum allowed probability, and a and b being the slope and intercept of the decay depended on $i_{\max}$.
- **Exponential decay**: $\varepsilon_i = k^i$ with $k < 1$ being a constant depending on $i_{\max}$.
- **Inverse sigmoid decay**: $\varepsilon_i = \frac{k}{k + \exp\left(\frac{i}{k}\right)}$ with $k \geq 1$ depending on $i_{\max}$.

The different decay schedules are visualized in Figure 3.11.

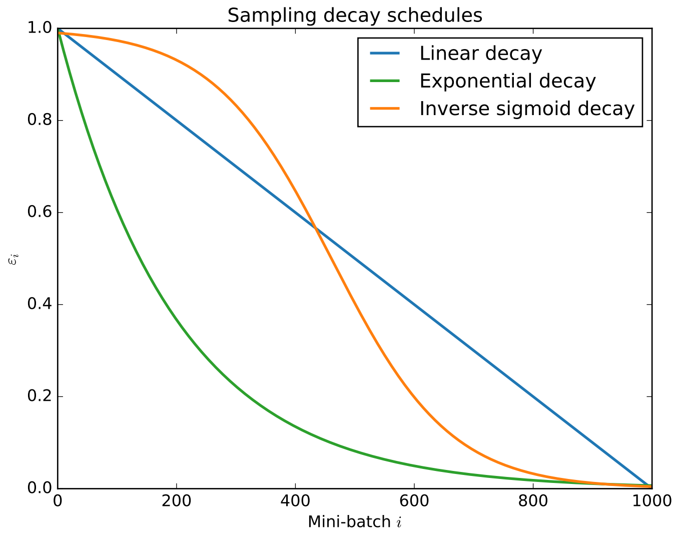


Figure 3.11: Example of the probability of using the true token as input at time-step t for a given mini-batch number i for different decaying schedules proposed in [4].

In this project, when Scheduled sampling was used, the Inverse sigmoid decay was used, as it was in [4].

CHAPTER 4

Results and discussion

In the following sections the most important results obtained in this project will be presented and discussed.

For all results in the following sections, except for the final results, the question and answer have been truncated to contain the first 100 characters in order to finish the runs within a fair amount of time. It is assumed that the corresponding results using full sequence lengths can be extrapolated from the obtained performance results of the truncated sequences, thus they form the basis of the decisions taken throughout the project.

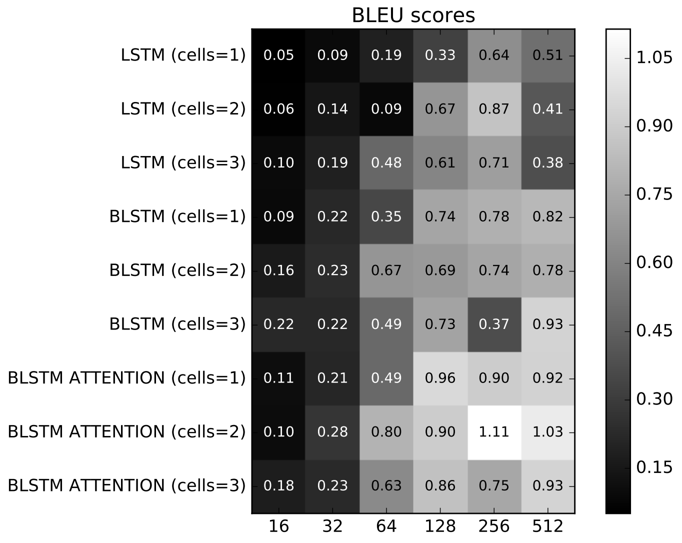
4.1 Optimal architecture

In order to decide on an optimal architecture for the network, the LSTM, BLSTM, and BLSTM Attention model was trained with different number of cells (number of recurrent layers), state sizes, character embedding sizes, and with and without the use of dropout on the Q/A dataset. Finally they were evaluated using the BLEU score.

The resulting BLEU scores can be seen in Figures 4.1 and 4.2.

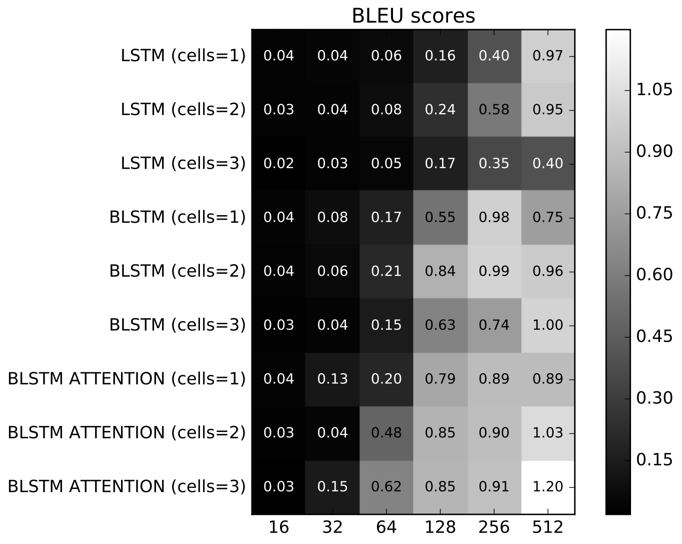
Note that in the following plots the state size referes to both the dimensionality of the state and the character embeddings.

Due to limited time and since statistical significant results are not required to get an idea of the optimal setup, the following results have not been cross-validated.



	16	32	64	128	256	512
LSTM (cells=1)	0.05	0.09	0.19	0.33	0.64	0.51
LSTM (cells=2)	0.06	0.14	0.09	0.67	0.87	0.41
LSTM (cells=3)	0.10	0.19	0.48	0.61	0.71	0.38
BLSTM (cells=1)	0.09	0.22	0.35	0.74	0.78	0.82
BLSTM (cells=2)	0.16	0.23	0.67	0.69	0.74	0.78
BLSTM (cells=3)	0.22	0.22	0.49	0.73	0.37	0.93
BLSTM ATTENTION (cells=1)	0.11	0.21	0.49	0.96	0.90	0.92
BLSTM ATTENTION (cells=2)	0.10	0.28	0.80	0.90	1.11	1.03
BLSTM ATTENTION (cells=3)	0.18	0.23	0.63	0.86	0.75	0.93

Figure 4.1: BLEU scores for the different models with varying number of cells and state sizes with no dropout.



	16	32	64	128	256	512
LSTM (cells=1)	0.04	0.04	0.06	0.16	0.40	0.97
LSTM (cells=2)	0.03	0.04	0.08	0.24	0.58	0.95
LSTM (cells=3)	0.02	0.03	0.05	0.17	0.35	0.40
BLSTM (cells=1)	0.04	0.08	0.17	0.55	0.98	0.75
BLSTM (cells=2)	0.04	0.06	0.21	0.84	0.99	0.96
BLSTM (cells=3)	0.03	0.04	0.15	0.63	0.74	1.00
BLSTM ATTENTION (cells=1)	0.04	0.13	0.20	0.79	0.89	0.89
BLSTM ATTENTION (cells=2)	0.03	0.04	0.48	0.85	0.90	1.03
BLSTM ATTENTION (cells=3)	0.03	0.15	0.62	0.85	0.91	1.20

Figure 4.2: BLEU scores for the different models with varying number of cells and state sizes with 50% dropout.

From inspecting Figures 4.1 and 4.2 it seems that the BLSTM Attention model is generally the better model. This is somewhat expected since this architecture achieves better performance on the SMT task.

Furthermore a higher BLEU score seems to be achieved when increasing the state size of the model. However when not applying any dropout during training it seems to have a sweet spot at 256 units. From inspecting the BLSTM Attention model's convergence plots, showing their training and validation loss during training, with and without dropout shown in Figures 4.3 and 4.4 it is seen that this might be caused by overfitting, since the validation loss for the models with 256 and 512 state sizes seem to be increasing when no dropout is used.

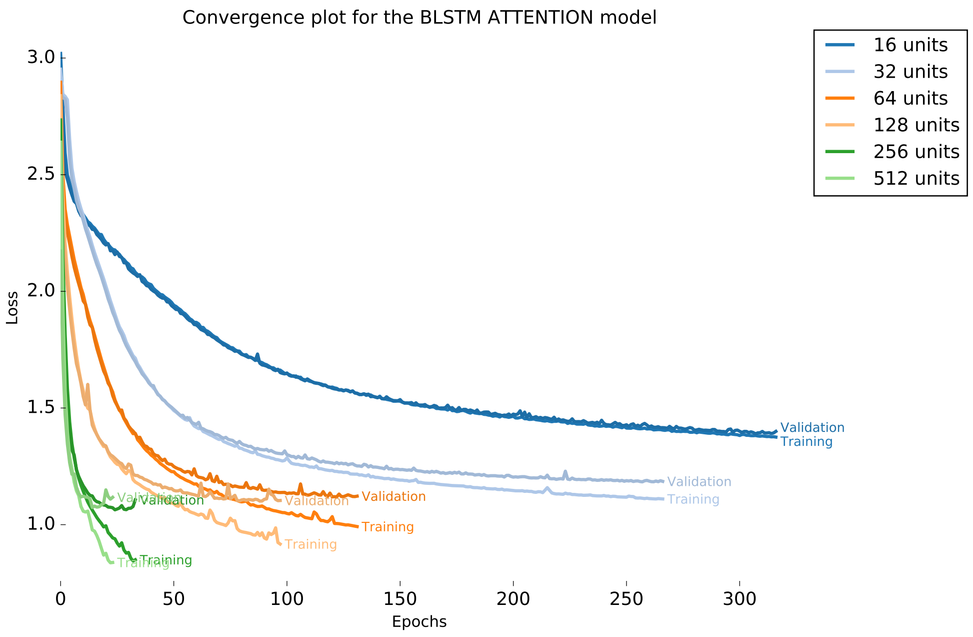


Figure 4.3: Convergence of the BLSTM Attention model without dropout using different number of units for the character embeddings and state size of the network.

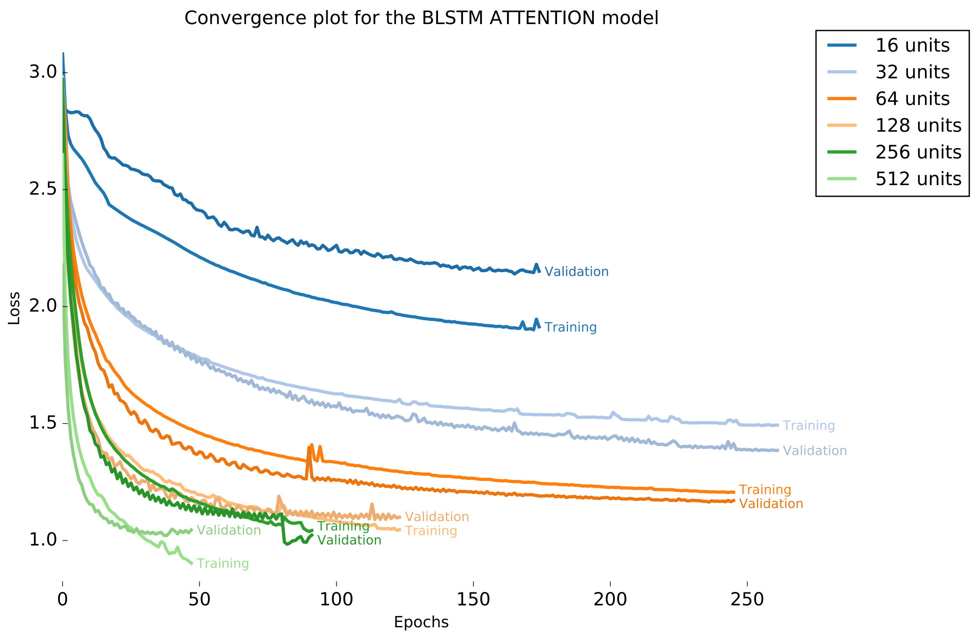


Figure 4.4: Convergence of the BLSTM Attention model with 50% dropout using different number of units for the character embeddings and state size of the network.

It is seen that when using dropout the models with small state sizes performs much worse than the corresponding models without dropout. From comparing the convergence plots of these models in Figures 4.3 and 4.4 it is also seen that the final validation loss reached when using dropout is higher than when not using it, hence using dropout on the less complex models have a negative effect on performance. This might be because of their lack of expressiveness due to their fewer number of parameters, hence disabling half of them during training makes it difficult for them to learn the underlying patterns.

From the convergence plots it is seen that when dropout is used, most of the models converge nicely. However the validation loss seem to consistently be very jagged, i.e. going up and down in small pikes. This could be a sign of a too high learning rate, hence decreasing the learning rate might improve performance, and is thereby done in the following test runs. Another general tendency seen is, that the more dimensions used for the state size and character embeddings, the faster the model converges (note that all the experiments shown in Figures 4.3 and 4.4 have used the same fixed learning rate).

Even though the BLSTM Attention model with 3 cells, state size of 512, and 50% dropout obtains the best BLEU score (1.20), the optimal architecture is chosen to be with 2 cells, state size of 256 and no dropout, which obtains a similar performance, but contains a lot fewer parameters.

All convergence plots for the models evaluated in Figures 4.1 and 4.2 can be seen in Appendix C.1.

4.2 Optimal encoding length

Since training time is highly dependent on the length of the encoded sequence, especially when attention is being used, the effect of the maximum encoding length on performance is tested. In Figure 4.5 the obtained BLEU scores for the BLSTM Attention model with different maximum encoding lengths trained on the Q/A dataset can be seen.

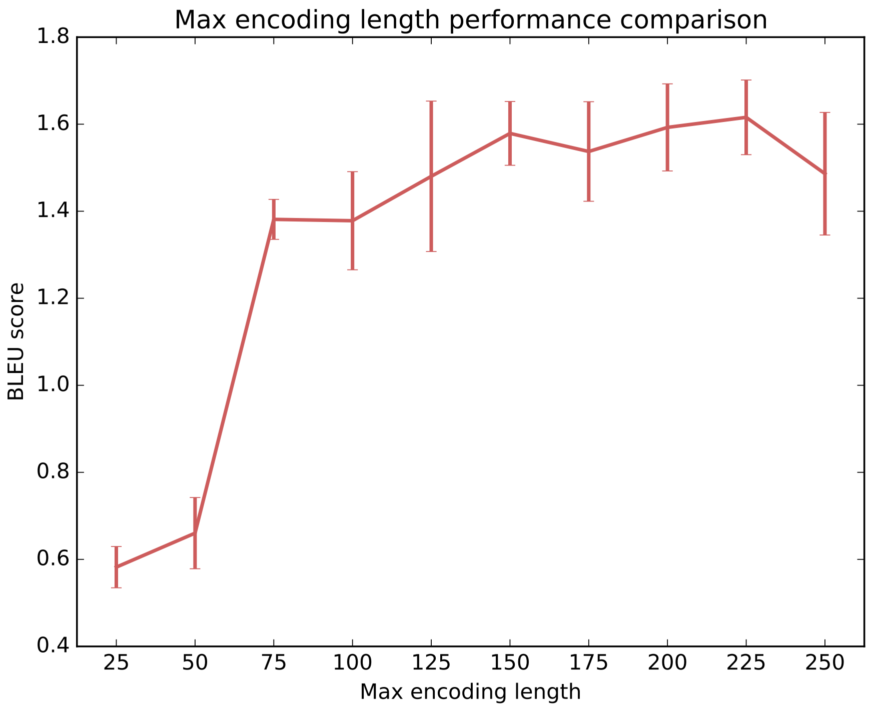


Figure 4.5: Cross-validated BLEU scores for the BLSTM Attention model with varying maximum encoding length (5-fold CV was used).

It is seen that the performance increase with the maximum encoding length. However increasing the encoding length above 200 seems to not have any significant effect. This

indicates that most information about the question is contained within the first 200 characters.
Even though better performance can be obtained with higher maximum encoding length the remainding runs will still use a maximum encoding length of 100 characters.

4.3 Optimal embedding size

Similarly as in Section 4.2 the performance effect of the embedding size, i.e. the dimensionality of the character embeddings, is examined. In Figure 4.6 the cross-validated performance of the BLSTM Attention model on the Q/A dataset with varying embedding size can be seen.

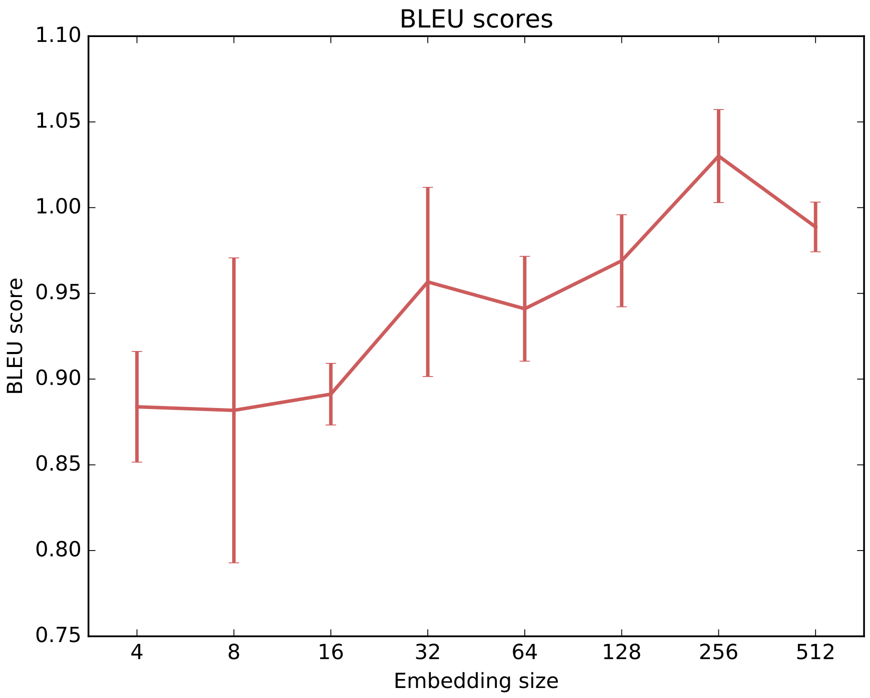


Figure 4.6: Cross-validated BLEU scores for the BLSTM Attention model with varying embedding sizes (3-fold CV was used).

From the plot it is seen that the performance again increase with the dimensionality, and that the optimal embedding size is found to be 256. One might believe an embedding size of 512 would obtain a performance at least as good as with 256, since the same amount of information can be held in each character vector, and even more.

The decay in performance might be due to the fact, that the increase in embedding size greatly increase the amount of connections between embeddings and hidden layers. This increases the importance of proper regularization of the network, which might havn't been done properly in this performance examination. I.e. the network might have overfitted the data when using an embedding size of 512.
In order to reduce the importance of proper regularization an embedding size of 256 is used in the remainding runs.

4.4 Dataset performance comparison

Due to the very limited size of the Q/A dataset (23.437 observations) the performance of the BLSTM Attention model was also evaluated on the Extended Q/A dataset (166.804 observations), and the subset of the Extended Q/A dataset coming from the HealthTap website which contributed with the most observations (137.052 observations).
The performance comparisons can be seen in Figure 4.7.

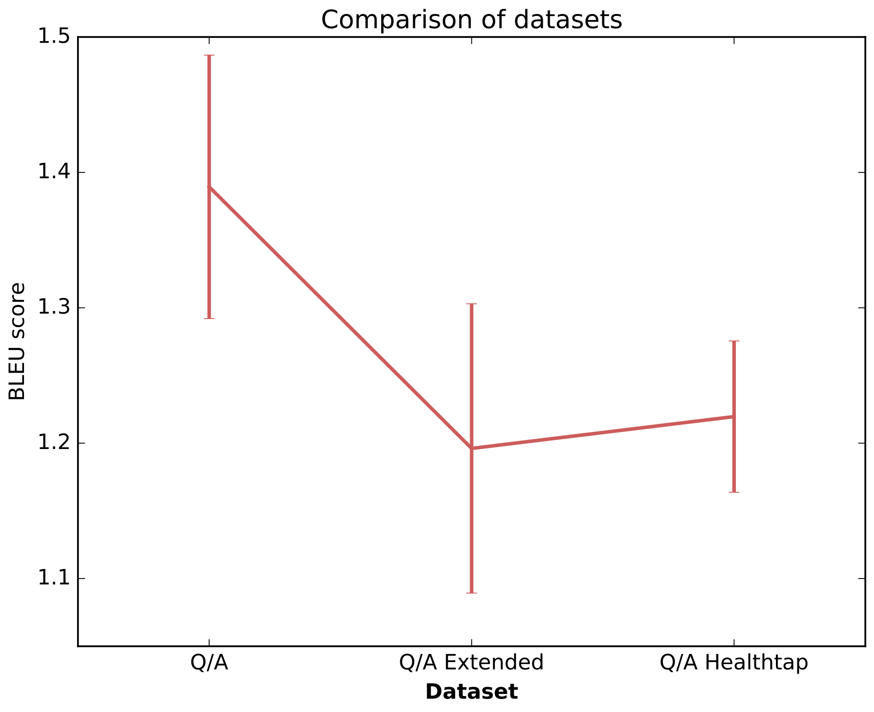


Figure 4.7: Cross-validated BLEU scores for the BLSTM Attention model with the three different datasets (3-fold CV was used).

Even though it is difficult to determine whether these three dataset contains Q/A's of same quality, and whether solving the STC task for each of them is of same difficulty, a comparison is still made. Based on the obtained BLEU scores and under the assumption that the Q/A's come from the same distribution, the Q/A dataset outperforms both the Q/A Extended and the Q/A Healthtap dataset, hence it is favoured as the dataset used for the remainding tests.

4.5 Scheduled sampling

In an attempt to solve the problem with maximum likelihood training explained in Section 3.6.1, the scheduled sampling approach explained in Section 3.6.2 was implemented and tested.

During training for each observation in a mini-batch and for each time-step t a coin toss with probability ε_i determines whether to use the true token as decoder input, or to use the token predicted at previous time-step $t-1$. The probability ε_i is decayed using the Inverse sigmoid decay explained in Section 3.6.2.

The convergence using scheduled sampling together with the probability ε_i can be seen in Figure 4.8

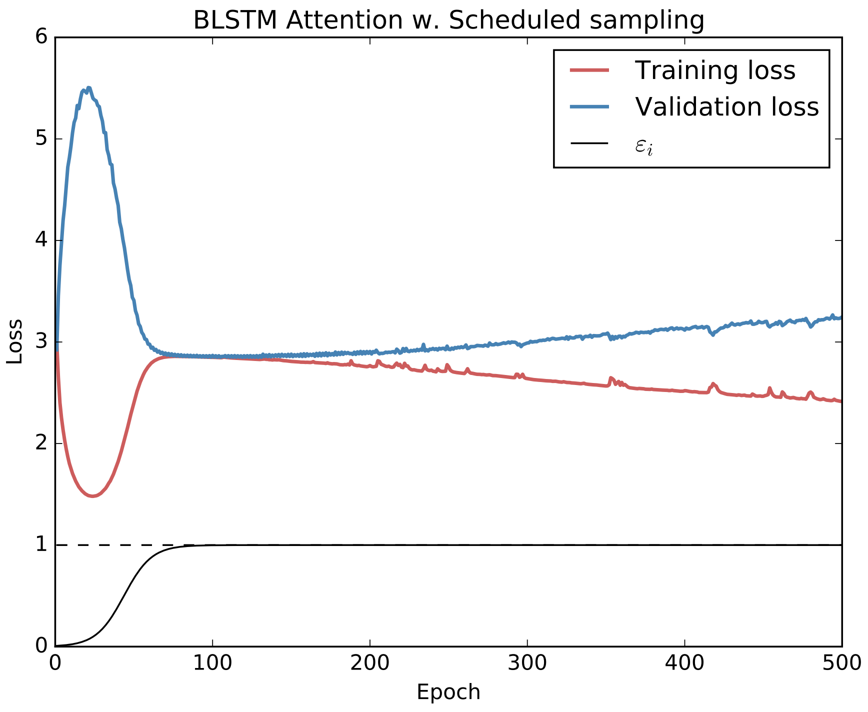


Figure 4.8: Example of convergence when using scheduled sampling. ε_i is the probability of using the true token at a given time-step for a given observation in a mini-batch during training, and is decayed using the Inverse sigmoid decay.

and as the plot indicates the model was not able to learn anything with this approach, even though the same approach as in [4] was used, hence it was dropped. Flipping the coin for the entire sequence instead of for each time-step, or flipping the coin for an entire mini-batch might yield other results, however this was not tested in this project.

4.6 Multi-task classifier

In order to explore the obtainable accuracy when classifying a questions category using the encoded state of the proposed models, a model without the decoder was trained with varying Q (size of dense layer taking the encoded vector as input explained in Section 3.3.7). I.e. an encoder predicting the question category was trained, and the optimal Q was found to be 128. Furthermore adding weight decay on the dense layers was found to improve performance slightly, with an optimal regularization parameter of $\lambda = 10^{-5}$.

The convergence and accuracy of the trained model can be seen in Figure 4.9.

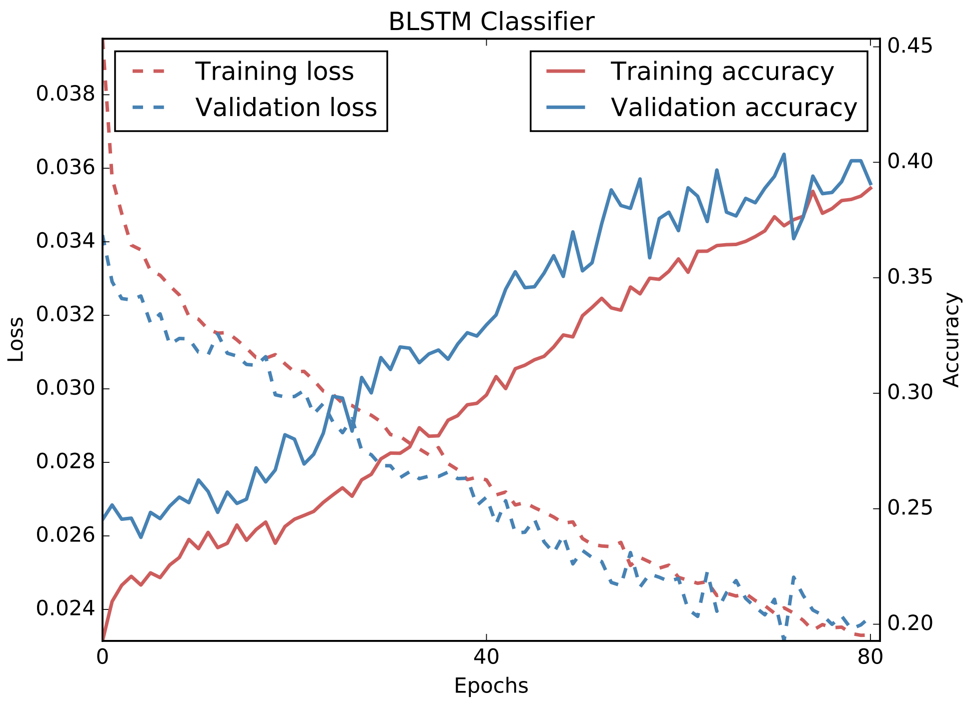


Figure 4.9: Training and validation loss and accuracy obtained using the encoder of the BLSTM model to classify question categories.

From the plot it can be seen, that a validation accuracy of approximately 40% should be possible for the multi-task network. Predicting the most frequent class every time yields an accuracy of approximately 25% (see Appendix B.1.1), hence 40% is better than the baseline but not impressive.

Extending the BLSTM Attention model to predict categories in the same way, generating answers simultaneously, a model refered to as the BLSTM Attention Classifier model is constructed.

In order to find the optimal mix of dropout and the α parameter (parameter weighing the sequence loss term vs. the category classifying term explained in Section 3.3.7), a grid search on the two hyperparameters were performed. The results can be seen in Figure 4.10

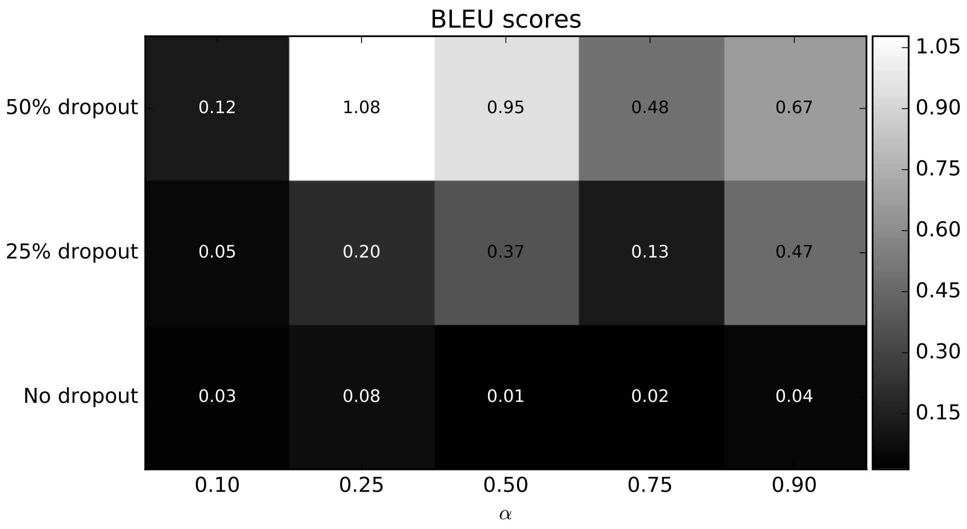


Figure 4.10: BLEU scores for the BLSTM Attention Classifier model using different α parameter and amount of dropout.

hence the optimal combination was found to be 50% dropout and $\alpha = 0.25$.

The convergence and accuracy of the model trained with the optimal values of dropout and α can be seen in Figure 4.11.

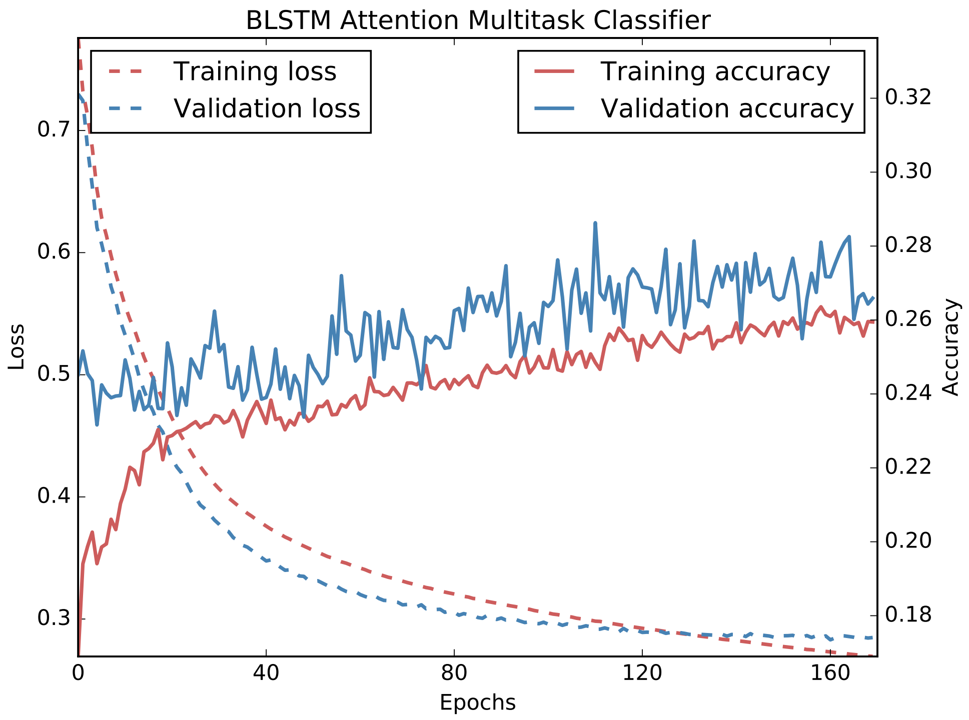


Figure 4.11: Training and validation loss and accuracy obtained using the BLSTM Attention Classifier.

Since the loss provided in Figure 4.11 consists of two loss terms defined in (3.4) the two terms are plotted side by side in Figure 4.12, where it is seen that the category classification loss is a little jagged, but the sequence loss is smooth and converges nicely.

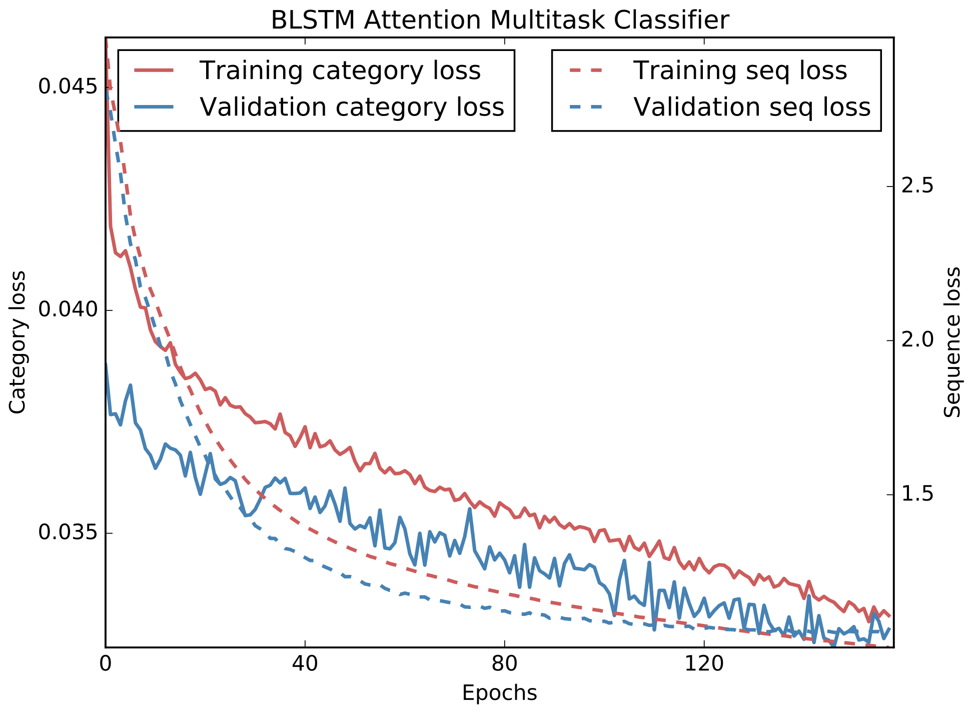


Figure 4.12: Convergence of the sequence loss and category loss terms for the BLSTM Attention Classifier.

4.7 Mixed LM

In order to determine the optimal mix ratio α_{mix} (fraction of Q/A observations used in each batch) when using a decoder with two modes, the BLSTM Attention model was trained as a mixed LM with different mix ratios and with and without dropout. The grid search result can be seen in Figure 4.13.

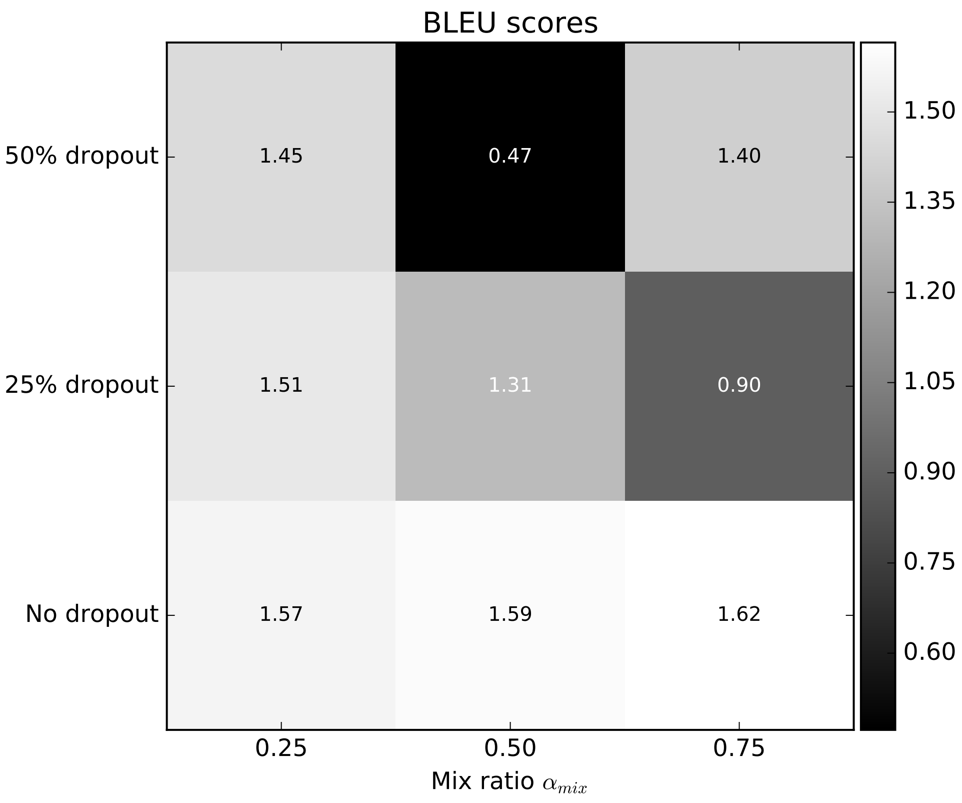


Figure 4.13: BLEU scores for the BLSTM Attention model with varying mix ratio and with and without dropout.

The optimal value is found to be a mix ratio of $\alpha_{\text{mix}} = 0.75$ and no dropout.

The same grid search was performed on the BLSTM Attention Classifier model where it is seen from Figure 4.14

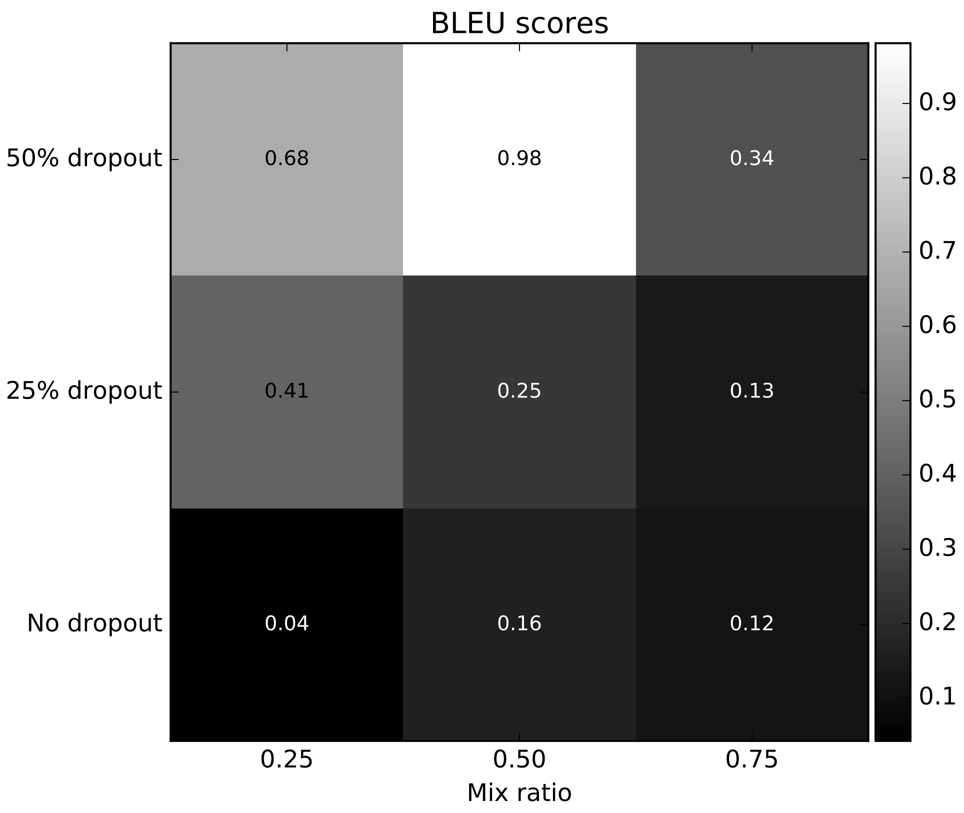


Figure 4.14: BLEU scores for the BLSTM Attention Classifier model using different mixing ratios and amount of dropout.

that the optimal mix ratio α_{mix} does not compare with the one found for the BLSTM Attention model, hence this hyperparameter is model specific thus it cannot be guaranteed that the optimal value of α_{mix} has been used in the remainding runs.

In Figure 4.15 the convergence of the BLSTM Attention model trained as a Mixed LM is compared to when trained regularly,

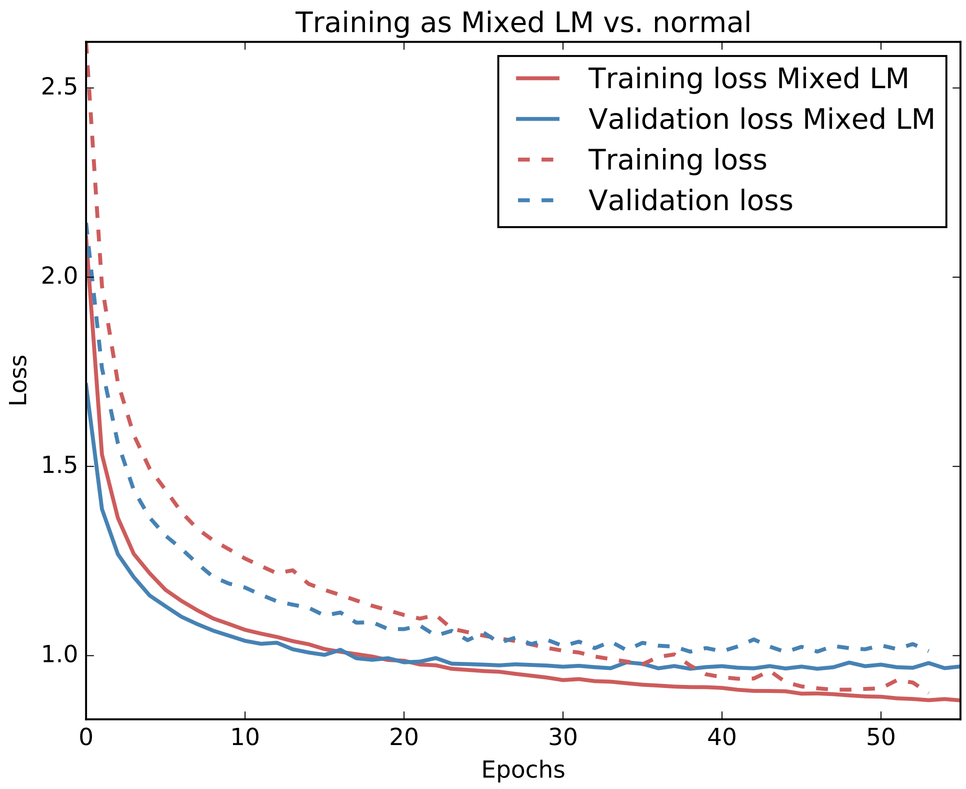


Figure 4.15: Convergence plot of the BLSTM Attention model trained as a Mixed LM vs. trained regularly.

where it is seen that training as a Mixed LM improves the loss slightly faster in the beginning, however the end result of the two approaches is very similar.
It was expected that the Mixed LM mode would regularize the network, however no sign of this is found.

4.8 Prediction strategies

The models trained in this project all learn distributions of characters in the given context, from which the next character is predicted. Utilizing this fact more sophisticated prediction methods can be applied, therefore the Beam Search method and Decaying sampling method explained in Sections 3.5.1 and 3.5.2.1 are tested and compared to the regular "*Argmax*" method (predicting the character with highest probability at each time-step).

Using Beam Search or Decaying sampling the optimal candidate is determined using the approach explained in Section 3.5.2.2 and used as the final answer prediction. The obtained accuracies of the trained Random Forest models are listed in Table 4.1

Model	Accuracy
RF_{question}	0.41 ± 0.01
RF_{answer}	0.31 ± 0.01

Table 4.1: Category classification accuracy (95% confidence interval) of the Random Forest models classifying question categories (RF_{question}) and answer categories (RF_{answer}).

where it is seen that the obtained accuracy on question category classification is slightly better than the one obtained using the BLSTM Classifier model based on the final state of the encoder. This shows that if the only purpose of classifying question categories in the network is to add information about the category to the decoder, one might as well use the predicted class probabilities from the RF_{question}. However in the hope of letting the network predict the category will result in the learned encoder state to contain more information about the category, this approach is kept throughout the project.

Comparing the performance of the "*Argmax*", Decaying sampling, and Beam search it is determined that using "*Argmax*" is better prediction strategy in this case. The results can be seen in Figure 4.16

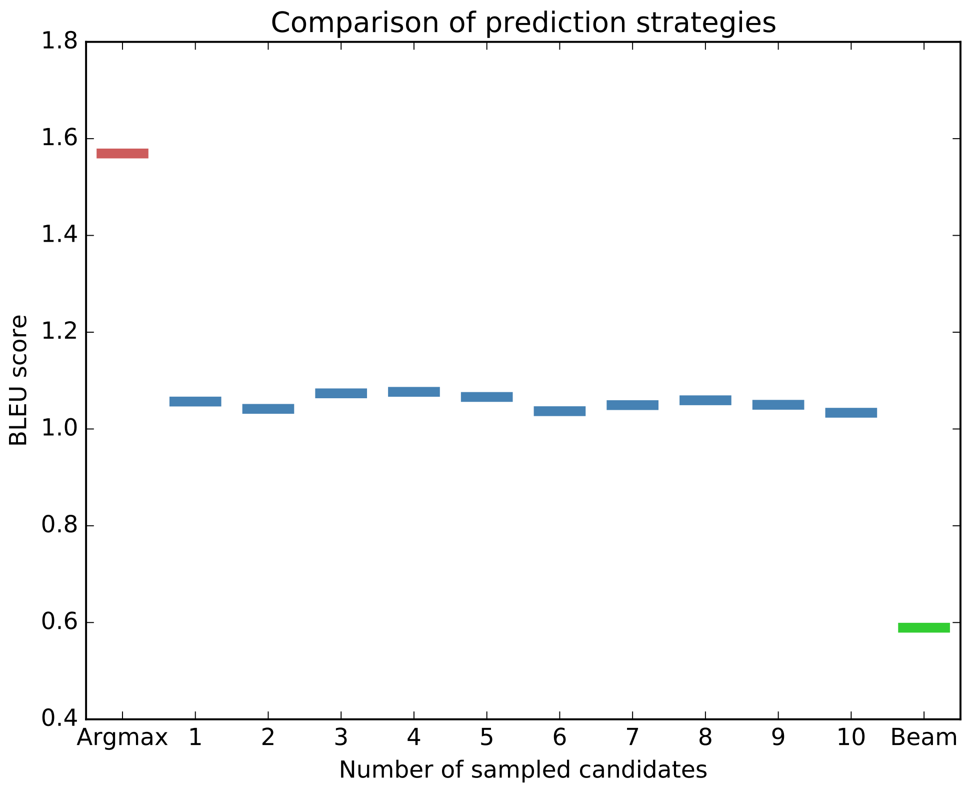


Figure 4.16: BLEU scores for the BLSTM Attention model comparing using the "*Argmax*" (red), Decaying sampling (blue) with varying sampling count B, and Beam search prediction strategy (green). A beam size of $B = 5$ was used for Beam search.

where red is the "*Argmax*" performance, blue is Decaying sampling with $B \in \{1, 2, \ldots, 10\}$, and green is Beam search with a beam size of $B = 5$.
It should be noted that the performance was not cross-validated, however a clear performance gap between the approaches can be seen from inspecting the plot, hence choosing "*Argmax*" as the optimal strategy seems fair.

4.9 Language model

The language model used for the Assisted LM models were trained using a single LSTM based RNN with 1 layer, 1024 state size, embeddings of size 256, and no dropout on the PubMed dataset with maximum sequence lengths of 500 characters. Training took approximately 2 weeks and the convergence plot can be seen in Figure 4.17.

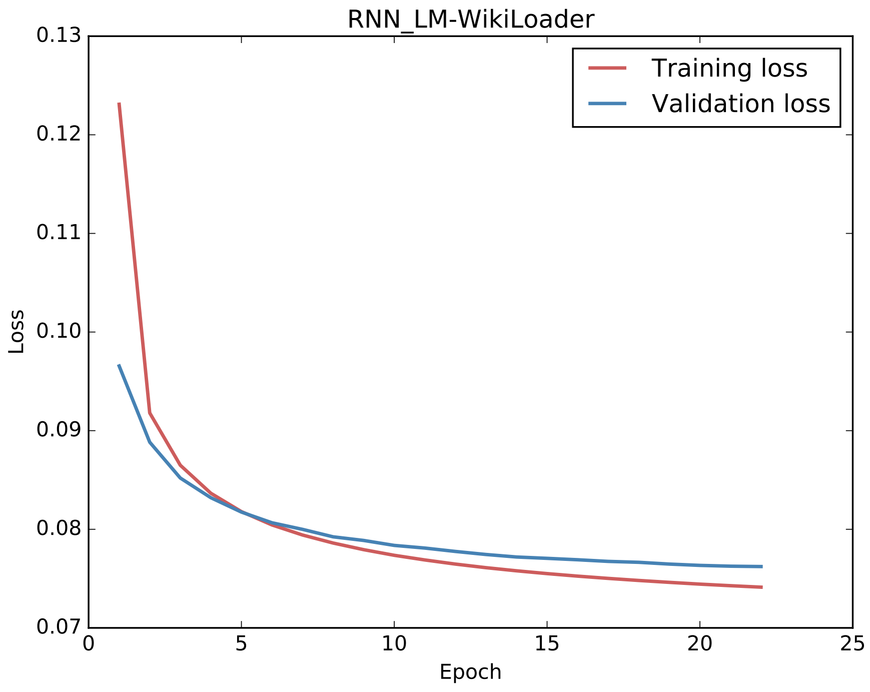


Figure 4.17: Convergence plot of the Language model trained on the PubMed dataset. Training was stopped after approximately 2 weeks.

In order to evaluate the quality of the features learned by the language model, a set of text samples were generated from sampling the character distribution learned by the model. A few examples can be seen in Table 4.2

Sample 1:	Lebsiella preservation was achieved as positive for cd49 in two cases of dens invaginatus and they measure 5___6 magnification.
Sample 2:	Vc increase to 4___2, which was reported in 27_ of 241 adult ecd____ (september 2008, 2013)
Sample 3:	El horses, bachelor_s chart, was observed after the appearance of spindle cells___m amount of alveolar mature conversion.
Sample 4:	Right-positive clinical stage was independent of greglochaghar et al d 23___m that prolapse by a muscle prolapse.
Sample 5:	Elling stored trials___men showed that bcr-based studies indicate that arrs are osteosarcomas to cough time___y.
Sample 6:	Letterfond undertaken, since lockote were fitted as fluorescence.
Sample 7:	Elling fractures with congenital anomalies of nose and spine mucogenotypes have been described.
Sample 8:	Ev weight and key may date would become interesting for patients because often associated to animal genders.

Table 4.2: Example of texts sampled from the language model.

It is seen that a lot of unknown symbols are predicted ("_"). This might indicate that the characters used for the alphabet might not be sufficient for the text in the PubMed dataset. Furthermore it is seen that not all sampled sentences make perfect sense, hence the network might need some more training time. However due to time limits this was not possible.

4.10 Prediction examples

In Tables 4.3 to 4.6 some examples of predictions on truncated Q/A pairs can be seen.

Question:	Can a former heart attack (19 yrs. ago) cause an abnormally on ekg when taking a stress test?
Answer:	Yes, an old heart attack may still show up on an electrocardiogram, especially when having a stress
Prediction:	Yes, but it is not a real time to treat any antibiotic that is related to the area of the time. An

Table 4.3: Q/A test sample prediction (question and answer truncated to 100 characters).

Question:	Is losing weight common while on on the depo provera shot?
Answer:	No. It is not common. Actually, a bit of weight.
Prediction:	No. It is not normal. It is not a really important to make sure that you are taking and will also

Table 4.4: Q/A test sample prediction (question and answer truncated to 100 characters).

Question:	What should i do if i suspect an overdose of humalog kwikpen?
Answer:	If overdose is suspected, contact your local poison control center or emergency room immediately. Us
Prediction:	If overdose is suspected, contact your local poison control center or emergency room immediately. Us

Table 4.5: Q/A test sample prediction (question and answer truncated to 100 characters).

Question:	Is it okay to have a tb skin test the day before oral surgery?. I need a tb test done for college an
Answer:	Yes. You can have a tb test done.
Prediction:	Yes, you can take a test for the severity of your self-diagnosis. Yhe only way to determine the su

Table 4.6: Q/A test sample prediction (question and answer truncated to 100 characters).

In Tables 4.7 and 4.8 some examples of predictions on full length Q/A pairs can be seen.

Question:	Can nuclear stress test cause cancer?
Answer:	No. It will not cause cancer. I had one myself.
Prediction:	Yes. According to the process can be caused by a problem.

Table 4.7: Example of contradicting Q/A test sample prediction.

Question:	How long after having unprotected sex can you find out you're pregnant. I had unprotected sex with my boyfriend 2-3 weeks ago and then 2 weeks later got the rod implant did a pregnancy test and it was negative but since I've had like 12 pees per day and it stings and tingles. Could I be pregnant?
Answer:	Usually about 2 weeks or more later.
Prediction:	Hi if you have been pregnant then the only way to get pregnant is a good idea to get pregnant and the only way to get pregnant if you have not had a pregnancy test done there its a bit of a big problem. Good luck

Table 4.8: Example of long and weird Q/A test sample prediction.

4.10.1 Learned template answers

From inspecting the test predictions it is found that the model has learned a few common answers, which seems to be some kind of template answers used by the professionals at WebMD.
In Table 4.9 an example of similar questions being answered with the same common answer, which has been learned by the model, can be seen.

Question 1:	What should I do if I suspect an overdose of agesic?
Question 2:	What should I do if I suspect an overdose of bidil?
Question 3:	What should I do if I suspect an overdose of verv?
Question 4:	What should I do if I suspect an overdose of cal-g?
Question 5:	What should I do if I suspect an overdose of copd?
Question 6:	What should I do if I suspect an overdose of desyrel?
⋮	⋮
Question 30:	What should I do if I suspect an overdose of albuterol?
Common answer:	If overdose is suspected contact your local poison control center or emergency room immediately. US residents can call the US national poison hotline at 1-800-222-1222. Canada residents can call a provincial poison control center.
Common predicted answer:	If overdose is suspected contact your local poison control center or emergency room immediately. US residents can call the US national poison hotline at 1-800-222-1222. Canada residents can call a provincial poison control center.

Table 4.9: Example of 30 similar test questions with a common template answer learned by the model.

In Table 4.10 a set of similar answers, which have been answered by the model with the same similar answer, can be seen.

Answer 1:	I don't have enough information (none in fact) about you to make a blind diagnosis. If you are having panic attacks you will need to see your medical provider or a psychiatrist to help you. There are some excellent medications that can hopefully fix this for you.
Answer 2:	It's impossible to say what this might be without performing a physical examination unfortunately. There are many structures such as lymph nodes and salivary glands in the area you mention. These could become swollen and painful or it could be something else entirely.
Answer 3:	I have no way of confirming or disputing your "allergic reaction" diagnosis but I can tell you that sulfa drugs can have a very dramatic rash as a non-allergic side-effect and this rash can last days. An allergic rash is typically very itchy hives. A sulfa drug reaction common toward the end of your course of treatment does not usually itch.
Answer 4:	Thank you for posting your question here at webmd answers. Your son's symptoms are too complicated to fully address here. Please keep working with your doctor.
Predicted answer:	I'm sorry to hear you're going through this. It sounds very uncomfortable. It is not possible to determine the cause of your symptoms and sometimes causing the production of an extremely colon.

Table 4.10: Example of a set of similar answers to given questions (questions not shown) all letting the user know they need more information, which have been given the same response with somewhat same message.

The fact that the model generates this answer to multiple different questions, which the professionals mean requires more information, indicates that the model have learned what type of questions usually is hard to answer online by a professional.

4.11 Europarl

In order to make sure the model performed properly on a well-known dataset the BLSTM Attention model was trained on the Europarl English-French dataset with source and target texts truncated to the first 100 characters.

Training took approximately 7 days and was done using piecewise decaying learning rate starting from 10^{-3} going to 10^{-6} reducing with a factor of 10 every time validation loss didn't improve in 3 epochs, which is seen in the convergence plot in Figure 4.18 as the little down-tick at epoch 22.

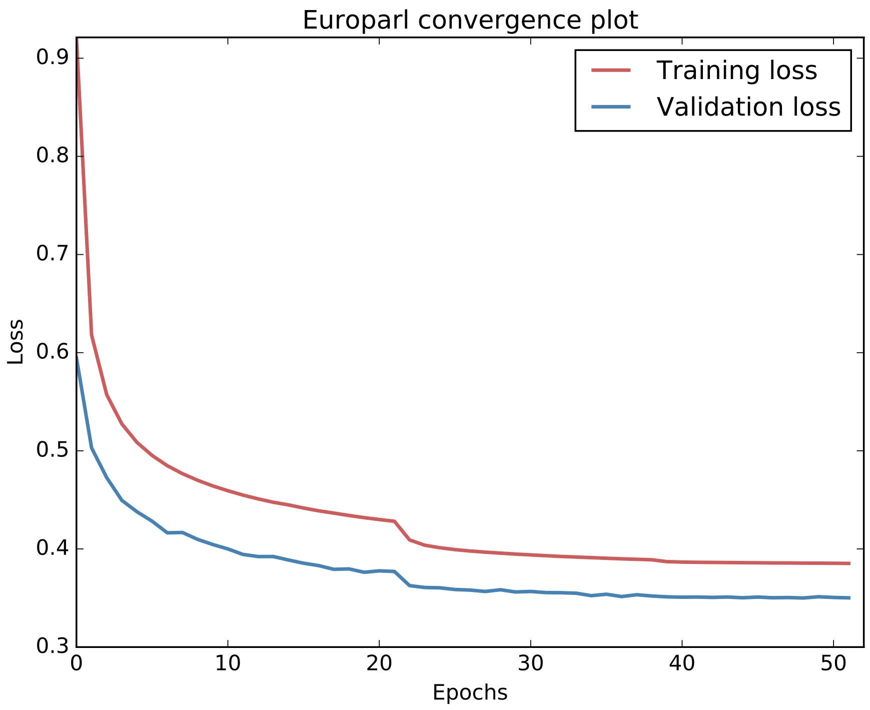


Figure 4.18: Convergence of the BLSTM Attention model on the Europarl English-French dataset.

The obtained performance is listed in Table 4.11.

Model	BLEU score
BLSTM Attention	13.27
State-of-the-art [35]	38.95

Table 4.11: Performance obtained on the Europarl English-French dataset using source and target texts truncated to the first 100 characters. Note the State-of-the-art result was obtained using full text lengths.

It is seen that the model performs far from the state-of-the-art on the Europarl English-French dataset, however it performs decent when taking into account that no optimization towards performace for this dataset have been done, hence it can be considered as an out-of-the-box performance.

In Tables 4.12 and 4.13 examples of a good and a bad translation can be seen.

English:	Secondly, I shall not vote for this commission because I do not believe that in real substance
French:	Deuxièmement, je ne voterai pas pour cette commission parce que, selon moi, rien, en substance, n'a
Prediction:	Deuxièmement, je ne voterai pas pour cette commission parce que je ne voterai pas pour cette

Table 4.12: Good translation test sample of of the Europarl English-French dataset.

English:	Firstly, we need to re-examine internal factors such as corruption, the lack of democratic
French:	D'abord, il faudra revoir les facteurs internes tels la corruption, le manque de garde-fous
Prediction:	Premièrement, nous devons réfléchir à la nécessité de réfléchir à la constitution de la commission

Table 4.13: Bad translation test sample of of the Europarl English-French dataset.

4.12 Final results

The final evaluation performance of the proposed models can be seen in Table 4.14 where they have been trained on the full length Q/A pairs.

Model	BLEU score	Loss	Category accuracy
LSTM	1.11	4.38	-
LSTM (Mixed LM)	0.54	3.45	-
LSTM (Assisted LM)	0.24	2.77	-
BLSTM	**1.76**	4.92	-
BLSTM (Mixed LM)	0.64	4.74	-
BLSTM (Assisted LM)	0.32	2.88	-
BLSTM Attention	1.05	4.46	-
BLSTM Attention (Mixed LM)	1.18	4.81	-
BLSTM Attention (Assisted LM)	0.30	2.76	-
BLSTM Attention C2W	1.51	4.79	-
BLSTM Attention C2W (Mixed LM)	1.34	4.99	-
BLSTM Attention C2W (Assisted LM)	0.46	2.91	-
Multi-task classifier	0.47	3.66	**0.38**
Multi-task classifier (Mixed LM)	0.38	4.07	0.36
Multi-task classifier (Assisted LM)	0.11	**2.49**	0.38

Table 4.14: Final performance on the Q/A dataset using the proposed models. Full length of questions and answers were used.

It is seen that final obtained test loss is generally much higher than the validation loss reported in the convergence plots. This is an effect of the problem with maximum likelihood training explained in Section 3.6.1. But it is also seen that the obtained BLEU scores are inverse proportional to the final loss, i.e. the more overfitted models obtain the better results. This might be caused by the amount of similar questions and answers in the data, hence making it favourable for the model to learn this small set of answers, and answer small variants of these for all questions. This is supported by the examples shown in Tables 4.9 and 4.10 and that from browsing the answers it was found that many variants of the listed common answers exist in the test predictions.

The Assisted LM models, where the predicted probabilities are merged with the probabilities of the language model, are found to decrease the final loss obtained, but not improve the BLEU score for any of the models. The only merging approach used was multiplying the probabilties, where the probabilities of the language model has a big impact, hence other approaches reducing this impact might have improved the performance, however this was not attempted. From the text samples generated by the language model seen in Table 4.2 it is also seen, that the language model could have been trained better. Training the language model on the Q/A dataset instead of the PubMed dataset might have made the language model learn character distributions more suitable for assisting the answer generating models, but this wasn't investigated either.

Since the set of valid answers to a given question can be very big and diverse compared to the source/translation pairs in Bilingual SMT, the BLEU score might not be a valid performance metric for this task. One could easily come up with a completely valid answer to a question, which would yield a BLEU score of 0 given the "*true*" prediction from the dataset considered in this project, hence another evaluation criteria is needed. In [28, 30] the predicted answers are evaluated manually by a set of humans who determines whether or not the given answers are valid. Due to lack of available resources this was not a possible solution in this project unfortunately, hence the use of the BLEU score. Furthermore the datasets used in [28, 30] contains observations in the order of millions which seems to be the required amount of data to learn proper translations in SMT, hence this is definitely also a significant lack in this project.

Recent research within Multi-task learning have shown that having a network learn multiple objectives simultaneously reduces the amount of data required to train the network, hence the Multi-task classifier could have made the lack of training data less of a problem, however from the obtained performance this cannot be concluded.

The number of learned parameters in the final models are listed in Table 4.15,

Model	# of parameters
LSTM	$1,640,208$
BLSTM	$2,690,832$
BLSTM Attention	$3,477,776$
BLSTM Attention C2W	$4,528,400$
Multi-task classifier	$4,800,400$
Language model	$5,246,976$

Table 4.15: Number of parameters in the final models.

and it is noted that the training time increase significantly with the increase in number of model parameters, ranging from a training time of approximately 10hours for the LSTM model trained on full sequence lengths to a training time of approximately 4 days for the BLSTM Attention C2W model. The Language model was trained on the PubMed dataset hence its training time cannot be compared to the others.

CHAPTER 5

Conclusion

From the study of this project it can be concluded that training an end-to-end network, using the Encoder-decoder framework similar to the one used for SMT, for generating answers to medical questions is a very difficult task, however somewhat possible. It has been shown that it is possible for a bidirectional recurrent neural network, with a final obtained BLEU score of 1.76, to learn certain fixed template answers to specific type of questions, and also in some way learn responses to specific types of questions requiring more information from the user. Furthermore a large fraction of the generated answers are correctly spelled, understandable answers, hence the model have learned some understanding of syntax and semantics.

However it must be stated that these answers have been manually picked from the test samples, and in most cases the answers generated by the model did not answer the question in any useful way. Often the model also end up in loop-like situations repeating itself multiple times.

Using a language model trained on general medical/health-related text to assist the answer generation for the proposed models, by merging their probabilities in each time-step, was found to not improve the quality of the generated answers.

Furthermore it has been found that the BLEU score is not a suitable performance evaluation metric for the STC problem, since the set of possible valid answers to a given question is much bigger and diverse than the one for Bilingual translation.

The proposed sampling techniques "*Decaying sampling*" and "*Beam search*" was found to not improve model performance despite its proved usefulness within SMT.

No results showed any evidence of that training a network partly as a language model and partly as an answer generating model had any improvements on the final performance. The early epochs showed some slight improvements in convergence speed, however each epoch was significantly slower due to the extra data and the decoder handling the "*mode-switching*" between language model-mode and answer generating mode.

Finally it was found possible to construct a model classifying the question category and generating answers simultaneously, however the obtained classification accuracy slightly decreased when the network had to predict sequences also. This might have been due to a lack in number of parameters of the model, however investigating this has been left for future work.

APPENDIX A

Theory appendix

A.1 Derivation of linear least-squares solution

Given training data consisting of inputs $\mathrm{X} \in \mathbb{R}^{(p+1)\times N}$ and outputs $\mathrm{Y} \in \mathbb{R}^{M\times N}$ the "*Residual Sum-of-Squares*" estimate of the weight matrix $\hat{\mathrm{W}}_{\mathrm{RSS}} \in \mathbb{R}^{M\times(p+1)}$ can be found by finding the roots of the derivative of the objective function

$$\mathrm{RSS}(\mathrm{W}) = \mathbf{1}^T(\mathrm{Y}-\mathrm{WX})^T(\mathrm{Y}-\mathrm{WX})\mathbf{1}. \tag{A.1}$$

I.e. from solving

$$\frac{\partial \mathrm{RSS}(\mathrm{W})}{\partial \mathrm{W}} = 0 \tag{A.2}$$

By removing the factors of one and expanding the objective function we find

$$\begin{aligned}
\mathbf{1}^T(\mathrm{Y}-\mathrm{WX})^T(\mathrm{Y}-\mathrm{WX})\mathbf{1} &= (\mathrm{Y}-\mathrm{WX})^T(\mathrm{Y}-\mathrm{WX}) && \text{(A.3)}\\
&= (\mathrm{Y}^T-\mathrm{X}^T\mathrm{W}^T)(\mathrm{Y}-\mathrm{WX}) && \text{(A.4)}\\
&= \mathrm{Y}^T\mathrm{Y}-\mathrm{Y}^T\mathrm{WX}-\mathrm{X}^T\mathrm{W}^T\mathrm{Y}+\mathrm{X}^T\mathrm{W}^T\mathrm{WX} && \text{(A.5)}\\
&= \mathrm{Y}^T\mathrm{Y}-\mathrm{Y}^T\mathrm{WX}-\mathrm{Y}^T\mathrm{WX}+\mathrm{X}^T\mathrm{W}^T\mathrm{WX} && \text{(A.6)}\\
&= \mathrm{Y}^T\mathrm{Y}-2\mathrm{Y}^T\mathrm{WX}+\mathrm{X}^T\mathrm{W}^T\mathrm{WX} && \text{(A.7)}
\end{aligned}$$

Computing the derivate leads to

$$\begin{aligned}
\frac{\partial \mathrm{RSS}(\mathrm{W})}{\partial \mathrm{W}} &= \frac{\partial\left(\mathrm{Y}^T\mathrm{Y}-2\mathrm{Y}^T\mathrm{WX}+\mathrm{X}^T\mathrm{W}^T\mathrm{WX}\right)}{\partial \mathrm{W}} && \text{(A.8)}\\
&= -2\mathrm{Y}^T\mathrm{X}+2\mathrm{X}^T\mathrm{W}^T\mathrm{X} && \text{(A.9)}
\end{aligned}$$

Setting the derivate equal to zero and solving obtains us

$$\begin{aligned}
-2\mathrm{Y}^T\mathrm{X}+2\mathrm{X}^T\mathrm{W}^T\mathrm{X} &= 0 && \text{(A.10)}\\
\mathrm{X}^T\mathrm{W}^T\mathrm{X} &= \mathrm{Y}^T\mathrm{X} && \text{(A.11)}\\
\mathrm{X}^T\mathrm{W}^T\mathrm{X}\mathrm{X}^T &= \mathrm{Y}^T\mathrm{X}\mathrm{X}^T && \text{(A.12)}
\end{aligned}$$

Since XX^T is a square matrix the inverse can be computed (assuming the matrix is not singular)

$$\begin{aligned}
\mathrm{X}^T\mathrm{W}^T\mathrm{XX}^T(\mathrm{XX}^T)^{-1} &= \mathrm{Y}^T\mathrm{XX}^T(\mathrm{XX}^T)^{-1} && \text{(A.13)}\\
\mathrm{X}^T\mathrm{W}^T &= \mathrm{Y}^T && \text{(A.14)}\\
\mathrm{XX}^T\mathrm{W}^T &= \mathrm{XY}^T && \text{(A.15)}\\
(\mathrm{XX}^T)^{-1}\mathrm{XX}^T\mathrm{W}^T &= (\mathrm{XX}^T)^{-1}\mathrm{XY}^T && \text{(A.16)}\\
\mathrm{W}^T &= (\mathrm{XX}^T)^{-1}\mathrm{XY}^T && \text{(A.17)}\\
\mathrm{W} &= \mathrm{YX}^T(\mathrm{XX}^T)^{-1} && \text{(A.18)}
\end{aligned}$$

where we have used the rules

$$\begin{aligned}
\left(\mathrm{A}^T\mathrm{BC}\right)^T &= \mathrm{C}^T\mathrm{B}^T\mathrm{A} && \text{(A.19)}\\
\left(\left(\mathrm{A}^T\mathrm{A}\right)^{-1}\right)^T &= \left(\left(\mathrm{A}^T\mathrm{A}\right)^T\right)^{-1} = \left(\mathrm{A}^T\mathrm{A}\right)^{-1} && \text{(A.20)}
\end{aligned}$$

A.2 Relation between sigmoid and tanh

The hyperbolic tangent $\tanh(x)$ and the sigmoid function σ given as

$$\text{Sigmoid :} \qquad \sigma(x) = \frac{1}{1+e^{-x}} \tag{A.21}$$

$$\text{Hyperbolic tangent :} \qquad \tanh(x) = \frac{e^{2x}-1}{e^{2x}+1} \tag{A.22}$$

have the following relationship

$$\tanh(x) = 2\sigma(2x) - 1 \tag{A.23}$$

which is given by

$$\begin{aligned}
\tanh(x) &= \frac{e^{2x}-1}{e^{2x}+1} && \text{(A.24)}\\
&= \frac{1-e^{-2x}}{1+e^{-2x}} && \text{(A.25)}\\
&= \frac{1}{1+e^{-2x}} - \frac{e^{-2x}}{1+e^{-2x}} && \text{(A.26)}\\
&= \frac{1}{1+e^{-2x}} - \frac{e^{-2x}+1-1}{1+e^{-2x}} && \text{(A.27)}\\
&= \frac{1}{1+e^{-2x}} - \left(\frac{1+e^{-2x}}{1+e^{-2x}} - \frac{1}{1+e^{-2x}}\right) && \text{(A.28)}\\
&= \frac{2}{1+e^{-2x}} - 1 && \text{(A.29)}\\
&= 2\sigma(2x) - 1 && \text{(A.30)}
\end{aligned}$$

APPENDIX B Methods appendix

B.1 The Q/A categories

The different categories for the Q/A dataset comma-separated and sorted alphabetically:

Acquired abnormality, Activity, Amino acid, peptide, or protein, Anatomical abnormality, Animal, Antibiotic, Bacterium, Biologic function, Biomedical occupation or discipline, Biomedical or dental material, Bird, Body location or region, Body part, organ, or organ component, Body space or junction, Body substance, Body system, Cell component, Cell or molecular dysfunction, Chemical viewed functionally, Classification, Clinical attribute, Conceptual entity, Congenital abnormality, Daily or recreational activity, Diagnostic procedure, Disease or syndrome, Educational activity, Entity, Environmental effect of humans, Enzyme, Eukaryote, Finding, Fish, Food, Functional concept, Fungus, Gene or genome, Genetic function, Geographic area, Health care activity, Health care related organization, Hormone, Idea or concept, Immunologic factor, Indicator, reagent, or diagnostic aid, Individual behavior, Injury or poisoning, Intellectual product, Laboratory procedure, Mammal, Manufactured object, Medical device, Mental or behavioral dysfunction, Mental process, Natural phenomenon or process, Neoplastic process, Organ or tissue function, Organism function, Organization, Pathologic function, Patient or disabled group, Pharmacologic substance, Phenomenon or process, Physiologic function, Plant, Professional or occupational group, Qualitative concept, Quantitative concept, Regulation or law, Research activity, Sign or symptom, Spatial concept, Substance, Temporal concept, Therapeutic or preventive procedure, Tissue, Virus

B.1.1 Category class distribution

The distribution of the question category classes can be seen in Figure B.1.

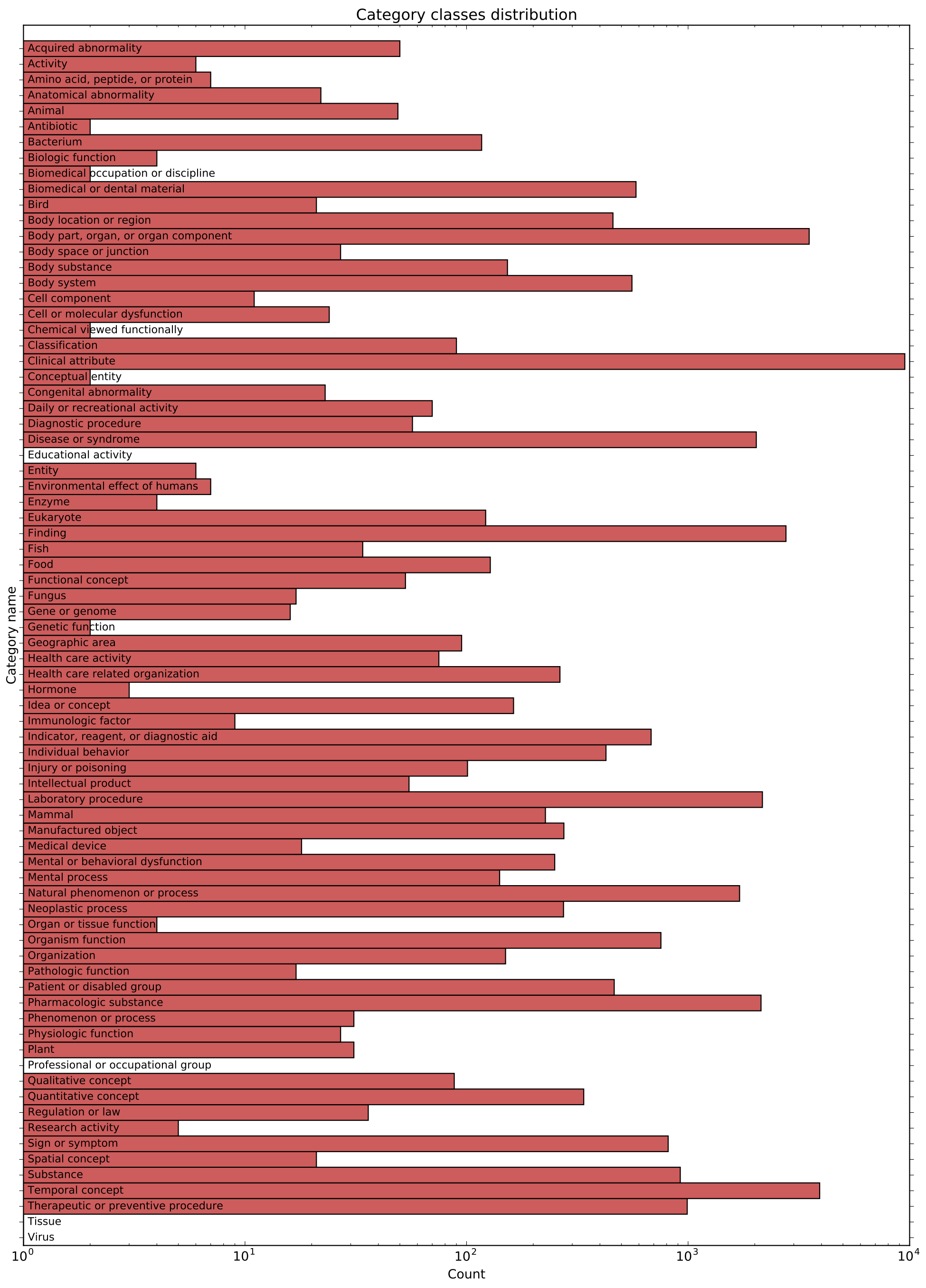


Figure B.1: Distribution of question category classes. Category occuring most times is "*Clinical attribute*" with 9510 occurencies, and the categories occuring least times are "*Virus*" and "*Tissue*" with only 1 occurence each.

B.2 Tensorflow example

A small `Tensorflow` example calculating the dot product of two vectors can be seen in Listing B.1.

```python

import numpy as np
import tensorflow as tf

x = tf.placeholder(tf.float64, shape=(10), name='x')
y = tf.placeholder(tf.float64, shape=(10), name='y')

z = tf.tensordot(x, y, axes=1)

with tf.Session() as sess:
    z_val = sess.run(
        fetches=[z],
        feed_dict={
            x: np.ones(10),
            y: np.ones(10) * 2
        }
    )
    print(z_val) # z_val = [20.0]
```

Listing B.1: Small `Tensorflow` example

APPENDIX C

Results appendix

C.1 Convergence plots

C.1.1 Without dropout

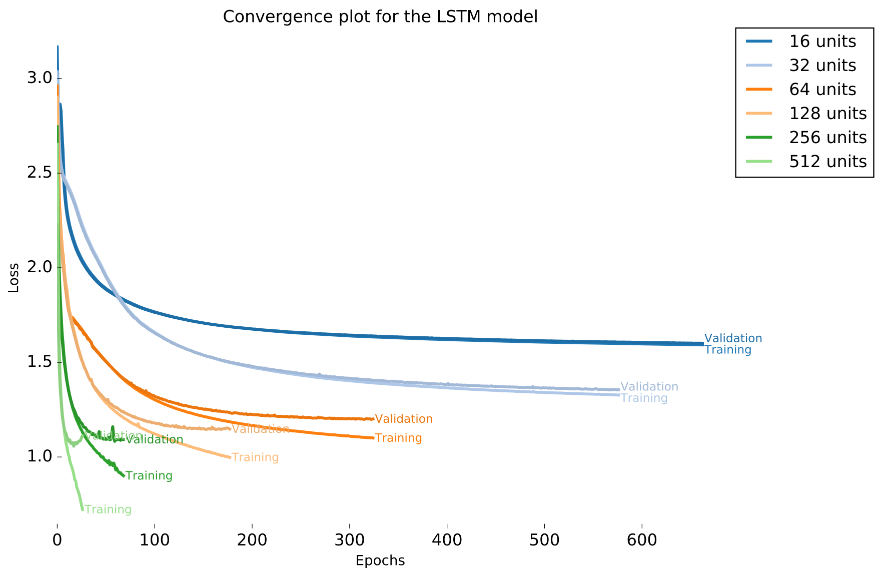


Figure C.1: Convergence plot of the LSTM model without dropout using different number of units for the character embeddings and state size of the network.

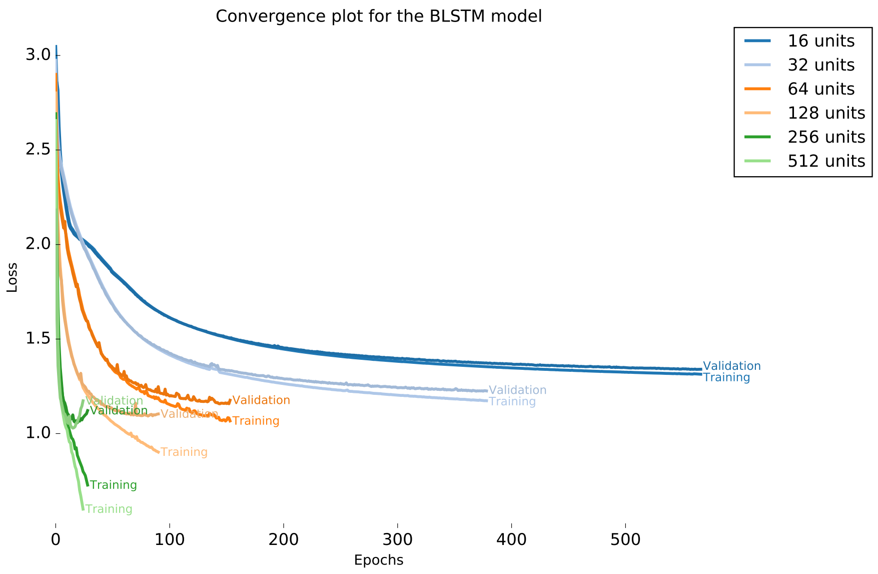


Figure C.2: Convergence plot of the BLSTM model without dropout using different number of units for the character embeddings and state size of the network.

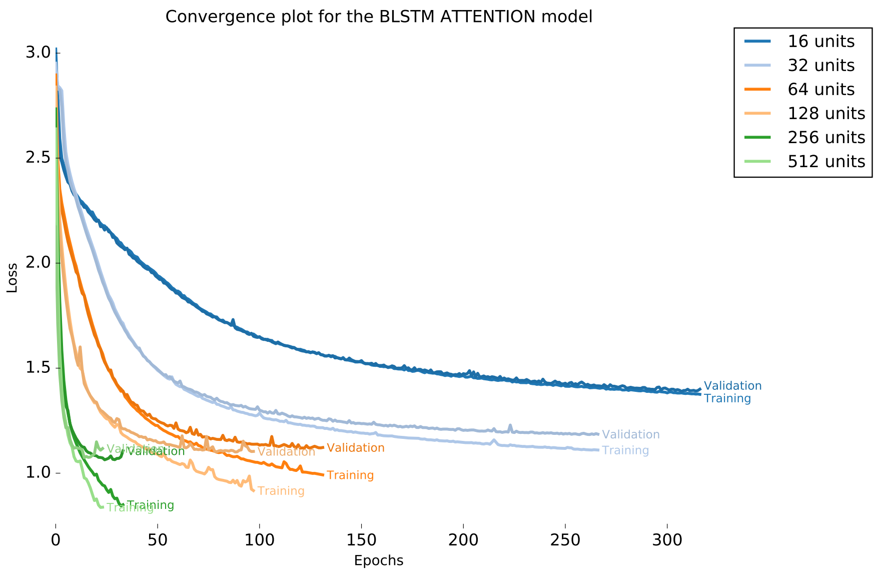


Figure C.3: Convergence plot of the BLSTM Attention model without dropout using different number of units for the character embeddings and state size of the network.

C.1.2 With dropout

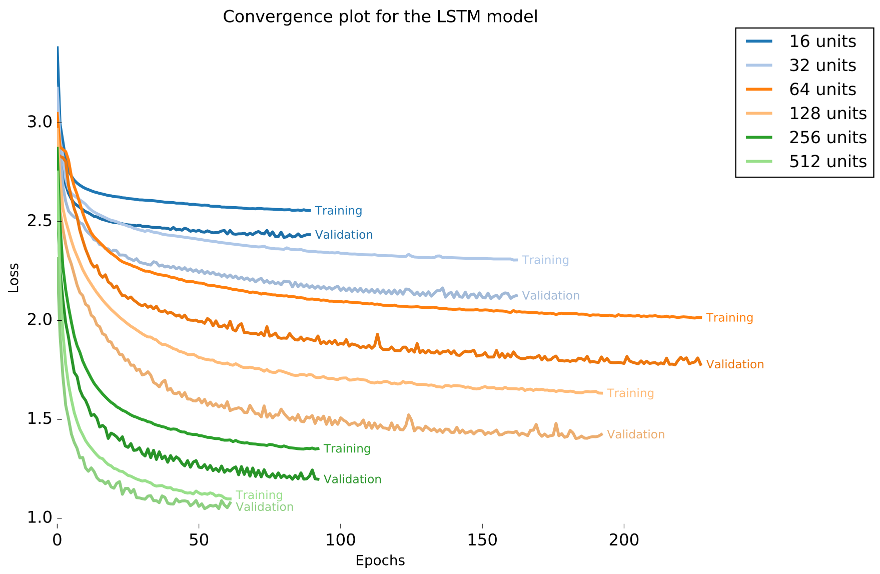


Figure C.4: Convergence plot of the LSTM model with 50% dropout using different number of units for the character embeddings and state size of the network.

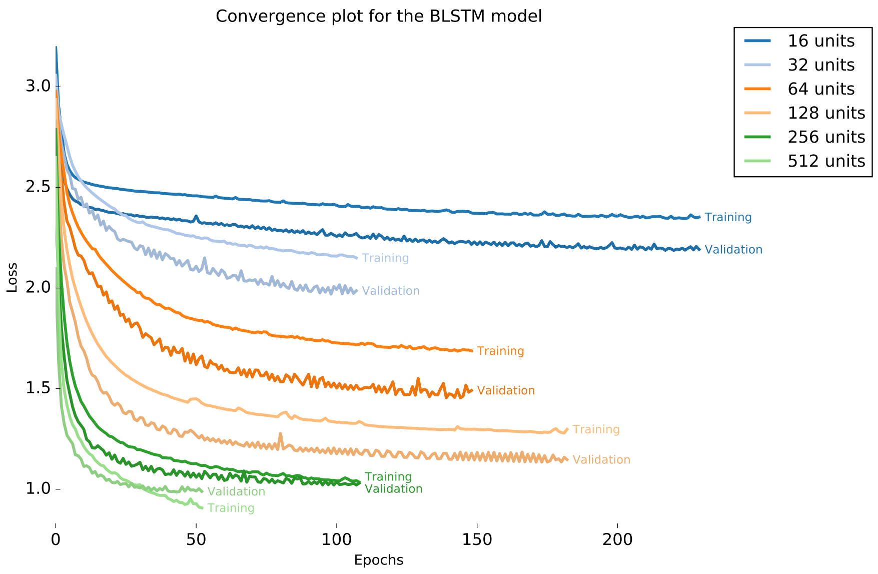


Figure C.5: Convergence plot of the BLSTM model with 50% dropout using different number of units for the character embeddings and state size of the network.

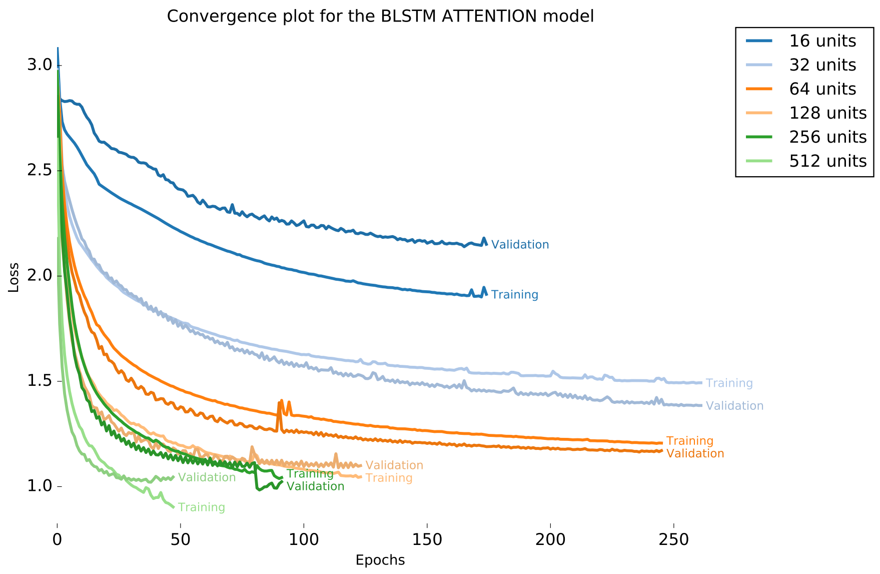


Figure C.6: Convergence plot of the BLSTM Attention model with 50% dropout using different number of units for the character embeddings and state size of the network.

APPENDIX D
Project process

D.1 Meeting summaries

The following subsections contain short summaries in bullet point form about the meetings held with FindZebra and Ole Winther throughout the project.

10th of January 2017

- Create document on slack with progress report.
- Introduction to SOLR
- Introduction to UMLS
- Introduction to the Flow framework

16th of January 2017

- Evaluation metrics: Edit distance, likelihood loss
- Look at paper by Alex Graves: "*Generating sequences with recurrent neural networks*"

30th of January 2017

- Try sampling from the softmax distribution to generate more diverse predictions
- Compare the C2W model with regular attention
- Dan has data from PubMed which can be used as extra training data for the language model
- Regularize the network

20th of February 2017

- Drop the C2W part of the encoder
- Make use of the question category in the network
- Try using t-SNE to on encoded questions to visualize encodings
- Try a bigger network
- Maybe generate multiple candidates and predict the optimal one

6th of March 2017

- Freeze the trained language model and use in prediction loop
- Implement Beam search
- Try out Tensorboard
- Implement a loss function with two terms, with one scaled with a hyperparameter

3rd of April 2017

- Try smaller cell state sizes, the network might be learning everything it can already
- Experiment with different maximum sequence lengths for encoder and decoder
- Take a look at layer normalization

18th of April 2017

- Try to plot distribution of sequence lengths when answer is truncated to contain the first 2-3 sentences
- Try to plot gradient norms in Tensorboard during training
- Read updated batch normalization paper

24th of April 2017

- Write project plan and progress report - hand in the 25th of April
- Maybe only remove duplicates in dataset when question AND answer is identical

8th of May 2017

- Check performance when varying max encoding sequence length
- Check if the extended dataset improves performance
- For training of the final models reduce learning rate when no improvement is found in order to squeeze the last performance out

22nd of May 2017

- Test the BLSTM Attention model on the Europarl English-French dataset
- Evaluate the model: Try and plot projections of encodings and check if similar sentences are close together

31st of May 2017

- Use true question class in decoder when predicting categories during training even though the network might be predicting something wrong.
- Sample which class a question comes from when a question have multiple classes
- If a class is not present pass the softmax values instead of a true class
- Send report to Ole
- Hand in the 27th of June instead of 3rd of July
- Remember to write a section about pre-processing of the data

15th of June 2017

- Include Europarl results in report
- Write about the language model and what was attempted

Bibliography

[1] Dario Amodei et al. “Deep Speech 2: End-to-End Speech Recognition in English and Mandarin”. In: *CoRR* abs/1512.02595 (2015). URL: `http://arxiv.org/abs/1512.02595`.

[2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. “Neural Machine Translation by Jointly Learning to Align and Translate”. In: *CoRR* abs/1409.0473 (2014). URL: `http://arxiv.org/abs/1409.0473`.

[3] Dzmitry Bahdanau et al. “End-to-End Attention-based Large Vocabulary Speech Recognition”. In: *CoRR* abs/1508.04395 (2015). URL: `http://arxiv.org/abs/1508.04395`.

[4] Samy Bengio et al. “Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks”. In: *CoRR* abs/1506.03099 (2015). URL: `http://arxiv.org/abs/1506.03099`.

[5] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006. ISBN: 0387310738.

[6] Kyunghyun Cho et al. “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”. In: *CoRR* abs/1406.1078 (2014). URL: `http://arxiv.org/abs/1406.1078`.

[7] Alex Graves. “Generating Sequences With Recurrent Neural Networks”. In: *CoRR* abs/1308.0850 (2013). URL: `http://arxiv.org/abs/1308.0850`.

[8] Alex Graves. *Supervised sequence labelling with recurrent neural networks*. Studies in Computational intelligence. Heidelberg, New York: Springer, 2012. ISBN: 978-3-642-24796-5. URL: `http://opac.inria.fr/record=b1133792`.

[9] Kazuma Hashimoto et al. “A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks”. In: *CoRR* abs/1611.01587 (2016). URL: `http://arxiv.org/abs/1611.01587`.

[10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001.

[11] Geoffrey E. Hinton et al. “Improving neural networks by preventing co-adaptation of feature detectors”. In: *CoRR* abs/1207.0580 (2012). URL: `http://arxiv.org/abs/1207.0580`.

[12] Hieu Hoang and Philipp Koehn. “Design of the Moses Decoder for Statistical Machine Translation”. In: *Software Engineering, Testing, and Quality Assurance for Natural Language Processing.* SETQA-NLP ’08. Columbus, Ohio: Association for Computational Linguistics, 2008, pages 58–65. ISBN: 978-1-932432-10-7. URL: `http://dl.acm.org/citation.cfm?id=1622110.1622120`.

[13] Sepp Hochreiter and Jürgen Schmidhuber. “Long Short-Term Memory”. In: *Neural Comput.* 9.8 (November 1997), pages 1735–1780. ISSN: 0899-7667. DOI: `10.1162/neco.1997.9.8.1735`. URL: `http://dx.doi.org/10.1162/neco.1997.9.8.1735`.

[14] K. Hornik, M. Stinchcombe, and H. White. “Multilayer Feedforward Networks Are Universal Approximators”. In: *Neural Netw.* 2.5 (July 1989), pages 359–366. ISSN: 0893-6080. DOI: `10.1016/0893-6080(89)90020-8`. URL: `http://dx.doi.org/10.1016/0893-6080(89)90020-8`.

[15] Heikki Hyyrö. *Explaining and Extending the Bit-parallel Approximate String Matching Algorithm of Myers.* Technical report. 2001.

[16] Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In: *CoRR* abs/1502.03167 (2015). URL: `http://arxiv.org/abs/1502.03167`.

[17] Zongcheng Ji, Zhengdong Lu, and Hang Li. “An Information Retrieval Approach to Short Text Conversation”. In: *CoRR* abs/1408.6988 (2014). URL: `http://arxiv.org/abs/1408.6988`.

[18] Alexander Rosenberg Johansen et al. “Neural Machine Translation with Characters and Hierarchical Encoding”. In: *CoRR* abs/1610.06550 (2016). URL: `http://arxiv.org/abs/1610.06550`.

[19] Diederik P. Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *CoRR* abs/1412.6980 (2014). URL: `http://arxiv.org/abs/1412.6980`.

[20] Philipp Koehn. “Europarl: A Parallel Corpus for Statistical Machine Translation”. In: *Conference Proceedings: the tenth Machine Translation Summit.* AAMT. Phuket, Thailand: AAMT, 2005, pages 79–86. URL: `http://mt-archive.info/MTS-2005-Koehn.pdf`.

[21] Guillaume Lample et al. “Neural Architectures for Named Entity Recognition”. In: *CoRR* abs/1603.01360 (2016). URL: `http://arxiv.org/abs/1603.01360`.

[22] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. 2015. URL: `http://tensorflow.org/`.

[23] Ishan Misra et al. “Cross-stitch Networks for Multi-task Learning”. In: *CoRR* abs/1604.03539 (2016). URL: http://arxiv.org/abs/1604.03539.

[24] Christopher Olah. *Understanding LSTM Networks*. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Online; accessed 17-April-2017]. 2008.

[25] Kishore Papineni et al. “BLEU: A Method for Automatic Evaluation of Machine Translation”. In: *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL ’02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pages 311–318. DOI: 10.3115/1073083.1073135. URL: http://dx.doi.org/10.3115/1073083.1073135.

[26] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “Understanding the exploding gradient problem”. In: *CoRR* abs/1211.5063 (2012). URL: http://arxiv.org/abs/1211.5063.

[27] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). URL: http://arxiv.org/abs/1506.02640.

[28] Alan Ritter, Colin Cherry, and William B. Dolan. “Data-driven Response Generation in Social Media”. In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. EMNLP ’11. Edinburgh, United Kingdom: Association for Computational Linguistics, 2011, pages 583–593. ISBN: 978-1-937284-11-4. URL: http://dl.acm.org/citation.cfm?id=2145432.2145500.

[29] M. Schuster and K.K. Paliwal. “Bidirectional Recurrent Neural Networks”. In: *Trans. Sig. Proc.* 45.11 (November 1997), pages 2673–2681. ISSN: 1053-587X. DOI: 10.1109/78.650093. URL: http://dx.doi.org/10.1109/78.650093.

[30] Lifeng Shang, Zhengdong Lu, and Hang Li. “Neural Responding Machine for Short-Text Conversation”. In: *CoRR* abs/1503.02364 (2015). URL: http://arxiv.org/abs/1503.02364.

[31] David Sussillo. “Random Walks: Training Very Deep Nonlinear Feed-Forward Networks with Smart Initialization”. In: *CoRR* abs/1412.6558 (2014). URL: http://arxiv.org/abs/1412.6558.

[32] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. “Sequence to Sequence Learning with Neural Networks”. In: *CoRR* abs/1409.3215 (2014). URL: http://arxiv.org/abs/1409.3215.

[33] Christian Szegedy et al. “Rethinking the Inception Architecture for Computer Vision”. In: *CoRR* abs/1512.00567 (2015). URL: http://arxiv.org/abs/1512.00567.

[34] Oriol Vinyals and Quoc V. Le. “A Neural Conversational Model”. In: *CoRR* abs/1506.05869 (2015). URL: http://arxiv.org/abs/1506.05869.

[35] Yonghui Wu et al. “Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation”. In: *CoRR* abs/1609.08144 (2016). URL: http://arxiv.org/abs/1609.08144.

[36] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. "Recurrent Neural Network Regularization". In: *CoRR* abs/1409.2329 (2014). URL: `http://arxiv.org/abs/1409.2329`.