BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

PUNCTUACTION RECONSTRUCTION FOR AUTOMATIC SPEECH TRANSCRIPTION

DOPLŇOVÁNÍ INTERPUNKCE DO AUTOMATICKÉHO PŘEPISU ŘEČI

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR — **TOMÁŠ ŠČAVNICKÝ**
AUTOR PRÁCE

SUPERVISOR — **Ing. IGOR SZÖKE, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2017

Abstract

This thesis deals with the problem of punctuation reconstruction in the output of automatic speech recognition systems. Constrains given on the solutions were applicability on general spoken English language and reasonable accuracy of the punctuation prediction system. Natural language tends to have in some cases non-deterministic nature and usually consists of a large number of grammatic rules. Therefore, a machine learning approach was chosen to solve this problem for its ability to recognize complicated patterns in data. A number of experiments with recurrent neural networks were executed to find the best network architecture for punctuation prediction. Resulting models created during these experiments reach accuracy comparable if not better than the works currently held as state-of-the-art solutions for punctuation reconstruction.

Abstrakt

Táto práca sa zaoberá rekonštrukciou interpunkcie vo výstupoch systémov na automatický prepis reči. Výsledný systém by mal byť schopný rekonštruovať interpunkciu vo všeobecnej zväčša hovorenej angličtine s rozumnou mierou presnosti. Prirodzený ľudský jazyk sa v istých prípadoch sa môže javiť nedeterministický a tvorba reťazcov často podlieha veľkému množstvu gramatických pravidiel. Kvôli tomu boli na predikciu interpunkcie vybrané algoritmy strojového učenia pre ich schopnosť rozoznať komplikované vzory v dátach. Bolo vykonaných niekoľko experimentov s rekurentnými neurónovými sieťami za účelom nájdenia najvhodnejšej architektúry modelu. Výsledné modely vytvorené počas týchto experimentov dosahujú presnosť porovnateľnú ak nie lepšiu než práce, v súčasnosti považované za najlepšie v obore.

Keywords

natural language processing, recurrent neural networks, machine learning

Klíčová slova

spracovanie prirodzeného jazyka, rekurentné neurónové siete, strojové učenie

Reference

ŠČAVNICKÝ, Tomáš. *Punctuaction reconstruction for automatic speech transcription.* Brno, 2017. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Szöke Igor.

Punctuaction reconstruction for automatic speech transcription

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Szőke. The supplementary information was provided by Mr. Szőke. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Tomáš Ščavnický
May 16, 2017

Acknowledgements

I would like to thank my supervisor Ing. Igor Szőke, Ph.D. and also Ing. Karel Beneš for help and guidance during my work on this project. I would also like to thank Bhargav Pulugundla for his insights from development of similar punctuation reconstruction systems.

Contents

Chapter 1

Introduction

Readable output of automatic speech recognition systems plays a huge role in their commercial adoption. When a system transcribes speech into text, users usually expect it to be properly segmented using correct punctuation. Besides the grammatical correctness it also increases readability of the text and contributes to the general experience user has with such a system. Many of the conventional automatic speech recognition system lack these features or their performance in reproducing punctuation is not satisfactory enough.

This thesis aims to address some solutions to this problem. It discusses a system for post-processing of the transcribed text. The mainly discussed approach used to process the transcribed text will be neural language modeling. Predominantly used neural networks in this paper will be recurrent neural networks which have shown to perform well on human language. Specifically a variation of recurrent neural networks called long short-term memory architecture will be used in the experiments due to its effectiveness. Long short-term memory networks are currently state of the art modeling technique for natural language and have found wide success in fields such as machine translation [24], question answering [26] and others. In later chapters results of experiments will be presented.

Besides choosing the right modeling techniques, implementation of the training pipeline will be described. Two different software frameworks were used during the process of development and qualities and drawbacks of each of them will be mentioned.

Chapter 2

Problem specification

The core problem of the assignment - punctuation prediction - is by all its means a very complicated process. The main language which will be evaluated is English language in which there are many grammatical rules and sentence structures dictating occurrence of punctuation in the written form of the language. These rules can be often combined into richer sentence structures and it is not unusual when the grammatical rules for punctuation are broken. A system for prediction of punctuation symbols needs to somehow embody these rules and be able to deterministically predict at each step whether the specific symbol should or should not be used.

Punctuation itself plays a crucial role in written language. It indicates indention of sentences or it can even give a rhythm to the sentence. Generally, punctuation works as some kind of additional indicator of metainformation about the written piece and gives it a sense of structure. Without this information the written piece can be interpreted by reader in a different way as were the author's intentions and it can become a lot harder for a reader to even comprehend the text and read through it. Therefore, experience of using automatic speech recognition systems without punctuation reconstruction can become unpleasing.

Since this problem is rather interesting, there have already been a couple of solutions in academic and commercial sphere. Academic attempts are represented by a number of papers such as following examples [9][21][8][18], though a paper which gave the most similar solution as the one proposed in this thesis is by Ottokar Tilk [25]. In this paper Mr. Tilk used long short-term neural networks on text processing with additional prosodic data indicating pauses in speech. With training data acquired from Estonian national radio Mr. Tilk's systems reached accuracy of approximately 0.7 F_1 score (on Estonian language). Systems developed for this thesis reach comparable score using only textual data. Commercial software such as Grammarly offers also systems able of rather accurate punctuation prediction, although almost no information about systems which Grammarly uses internally is public.

The problem of punctuation prediction can be broken down into evaluation of space between each two consequential words. The system also needs to somehow remember a relevant number of proceeding words and maybe even have information about succeeding words to have enough information about currently evaluated space between words. In attempt to recognize the underlying grammatic rules for punctuation, patter recognition algorithms, specifically neural networks were chosen.

Chapter 3

Neural networks for natural language processing

The beginning of language modeling with neural networks can be marked in the year of 1991 [11]. Firstly, feed forward neural networks (FFNNs) were used for language modeling. In the first decade of the new millennium, recurrent neural networks (RNNs) rapidly became popular among the scientific community surpassed FFNNs with the accuracy of modeling [23]. Andrej Karpathy's blog on language modeling using RNNs [6] gives many example of how effective RNNs can be.

3.1 Neural networks

Neural network is a mathematical model which transforms an input vector into an output vector using internally encoded functions. The motivation behind artificial neural networks is to simulate any function which transform x input data into y output data. It was firstly proposed by McCulloch and Pitts in 1943 [10]. In 1969 Marvin Minsky and Seymour Papert proposed a paper in which they proved that single-layer perceptron were incapable of processing the exclusive-or circuit [16]. This problem was, however, later resolved by adding hidden layers into neural networks.

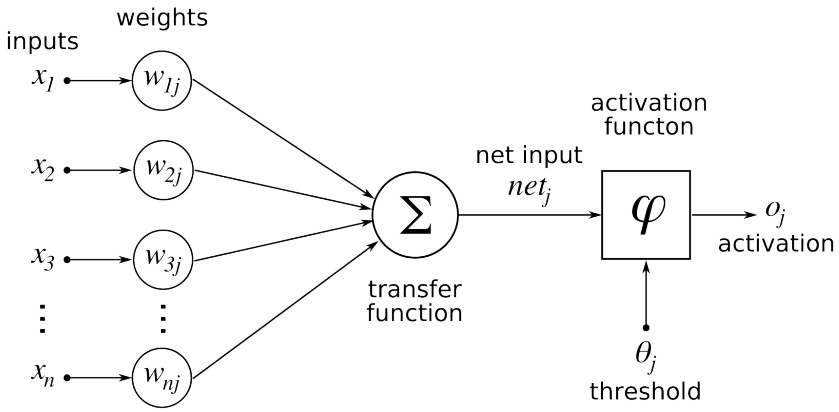


Figure 3.1: Model of basic artificial neuron

Basic topology of neural network consists of neurons which represent nodes of a graph. Neurons are defined by their weights w and bias b. Weights represent edges of the graph and connect the neurons. The final part of a basic neuron is activation function φ which calculates the final output of the node. Following formula mathematically describes behavior of a neuron.

$$y = \varphi(\sum_{i=1}^{I} w_i x_i + b) \tag{3.1}$$

On the higher level of abstraction, neurons in a neural network form two or more layers. A layer is a set of neurons. The first layer of a neural network is called the input layer and the last layer is called the output layer. Any layers in between are hidden layers. Signal in neural network is propagated from an input layer through hidden layers to an output layer. The forward propagation is carried by weights w which interconnect each neuron in two neighboring layers. In FFNNs, connections between layers have solely in one direction, i.e. output vector of the first layer is input vector of the second layer and so on. Image 3.1 describes basic structure of FFNN.

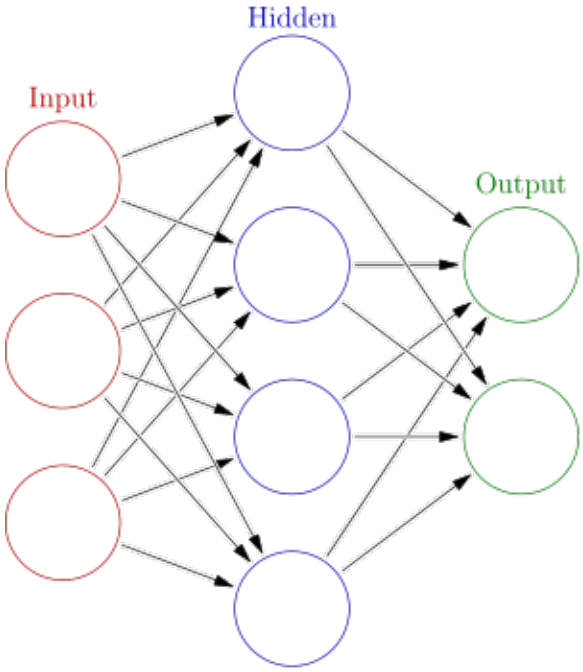


Figure 3.2: Feed forward neural network with one hidden layer

3.1.1 Training

The transformation of input vector into vector depends only on the weights and biases of the nodes in the network. We can elaborate this statement further and raise two questions:

1. How are the nodes initialized?
2. How can we adjust the values in the nodes so that we get the desired output?

There are numerous ways of initializing weights and biases in a neural network. The simplest and most straight forward solution is to initialize all the values to zero. However, this approach has proven to decrease learning rate and get the weight gradients stuck in local minimums [5]. Experiments have shown that far better training results are obtained when weights and biases are initialized according to normal distribution with zero mean value and variance of approximately 0.01 [5].

The feature of neural networks that brought the most attention to them is that they can learn.

3.1.1.1 Gradient Descent

In this section supervised learning for neural networks will be discussed. In supervised learning, a set of $(x, y); x \in X, y \in Y$ variables is given and we are training the network so that it would learn to simulate function g such that $g : X \to Y$. First, the training algorithm must use an objective probabilistic function to calculate the probability of the value $g(x)$ being correctly classified by the network as y. Here is an example of an objective probabilistic function for simple binary classification.

$$p(t \mid X) = \prod_n y_n^{t_n} (1 - y_n)^{1-t_n} \tag{3.2}$$

y_n is probability of class C predicted by the output of the neural network for input values x_n. t is a vector of correct class identities; $t = 0$ if x_n belongs to class C_1 and $t = 1$ if x_n belongs to class C_2. Our goal is to maximize the objective function 3.2. It is often more useful to work with the logarithm of the objective function. The following function is an example of what is commonly referred as *error function* or *cross entropy*, though this one is specific to the objective function 3.2. It is computed as a negative logarithm of the objective function (w represents the vector of weights of the network).

$$E(w) = -\ln(p(t \mid w)) = -\sum_{n=1}^{N} \{t_n \ln(y_n) + (1 - t_n) \ln(1 - y_n)\} \tag{3.3}$$

During the training we want to minimize the value of the error function (instead of maximizing the objective function, because it was negated). To accomplish this, gradient of error function is computed.

$$\nabla E(w) = \sum_{n=1}^{N} (y_n - t_n) x_n \tag{3.4}$$

When such a w is found that $\nabla E(w) = 0$, then the optimum of the error function was found. Yet, this may not be the best configuration of weights for the network, as will be mentioned later.

The optimum of the error function cannot be computed analytically, therefore it is computed numerically using following algorithm called *gradient descent*.

$$w^{\tau+1} = w^{\tau} - \eta \nabla E(w) \tag{3.5}$$

Following algorithm works fine with training linear regression systems, but since FFNNs are hierarchical linear regressions, the value of error function needs to be redistributed to the nodes of the neural network.

3.1.1.2 Backpropagation

To calculate the adjustment which should be made at a specific node, first the derivative of the error function with respect to the specific weights needs to be calculated using the chain rule. δ_k represents output error backpropagated to the layer k.

$$\frac{\partial E}{\partial w_{k,n}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial w_{k,n}} = \delta_k \cdot x_n \tag{3.6}$$

A general rule for calculating δ_k for neural network with M outputs. δ_m is output error on the output y_m.

$$\frac{\partial E}{\partial w_{k,n}} = \Big(\sum_{m=1}^{M} \delta_m w_{m,k}\Big) h_k(1-h_k) = \delta_k \tag{3.7}$$

After the error value for a node has been calculated, the node's weights can be updated using gradient descent algorithm 3.5.

3.2 Recurrent neural networks

Neural networks with feed-forward processing have great performance in some tasks, but they perform poorly when presented with sequential data. When given sequential data, we want our model to predict the next token in the series given the previous observations. In the context of language modeling this could mean predicting next character given all the previous characters, or as will be later mentioned predicting a whole word given previous words. This can be achieved by word embeddings.

What gives RNNs advantage over FFNNs is that FFNNs can use only a finite number of previous tokens to predict the next one while RNNs by their nature predict the next token using information from all the previous tokens [14].

The general topology of RNN consists of input layer, hidden layer with recurrent connections and output layer. Recurrent connections in the hidden layer are basically loops from the output of the hidden layer to its input. Therefore, when hidden layer in RNN is presented with data from RNN's input layer in time t_n it also takes in consideration its own state from the previous time step t_{n-1}. And since the previous time step also contains information about the hidden layer's state in t_{n-2} we can say that in this fashion, a RNN can hold in memory information about every data it has processed since t_0. The basic implementation of recurrent neural network is called Elman network after an American cognitive scientist [3].

Value of weights in the hidden layer of the network can be also called *state* of the network. It can be described by following equation where where s_t is the state of the network in time step t, U is vector of weights which are applied to the input vector and W is vector of recurrent weights.

$$s_t = \tanh(Ux_t + Ws_{t-1}) \tag{3.8}$$

State s_t is then run through vector V and softmax activation function to produce output o_t.

$$o_t = softmax(Vs_t) \tag{3.9}$$

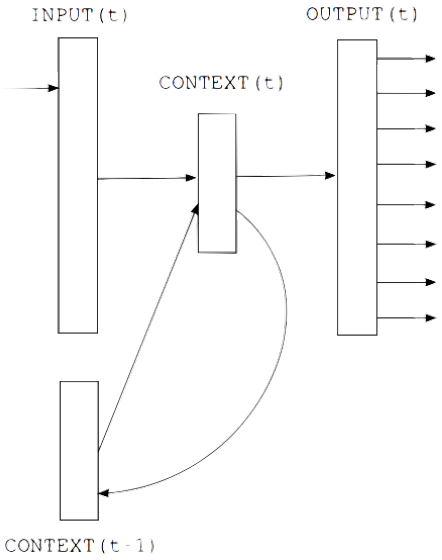


Figure 3.3: Scheme of recurrent neural network

3.2.1 LSTM

Long short-term neural network (LSTM) is a variation of RNN. The main difference is that they introduce a so called memory cell. Memory cell is a node of a network with LSTM architecture. Their architecture is more advanced that architecture of nodes in previously mentioned neural networks. The similarity with standard recurrent neural network is that form a chain from time t_n to t_0. However, LSTMs instead of having just one neural layer, have four interacting in specific ways.

The key element of an LSTM cell is the cell state. It is recurrently fed by the state the previous cell and current input data can modify this state by linear interactions. LSTM also allows information to flow freely through the cell state and not be modified by current input data. This is a huge advantage of LSTM architecture.

LSTM can modify its current state by so called gates. Altogether, there are three gates: forget gate, input gate and output gate. Forget gate decides what information from the state s_{t-1} should be abandoned. Input gate updates the current state and output gate calculated the output. All of the three gates work with current input vector x_t [17][4].

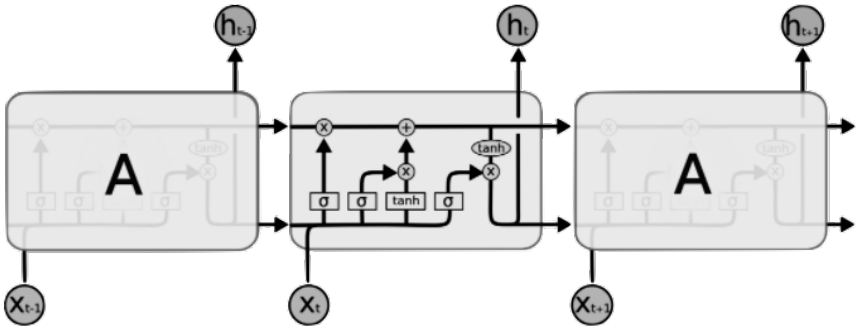


Figure 3.4: Vanilla LSTM architecture, courtesy of Christopher Olah [17]

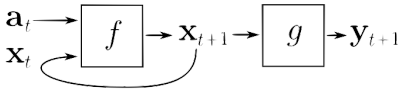
State, gates and output of the cell are mathematically described by following equations.

$$
\begin{aligned}
z^t &= g(W_z x^t + R_z y^{t-1} + b_z) && \text{cell input} \\
i^t &= \sigma(W_i x^t + R_i y^{t-1} + p_i \cdot c^{t-1} + b_i) && \text{input gate} \\
f^t &= \sigma(W_f x^t + R_f y^{t-1} + p_f \cdot c^{t-1} + b_f) && \text{forget gate} \\
c^t &= i^t \cdot z^t + f^t \cdot c^{t-1} && \text{cell state} \\
o^t &= \sigma(W_o x^t + R_o y^{t-1} + p_o \cdot c^{t-1} + b_o) && \text{output gate} \\
y_t &= o_t \cdot h(c^t) && \text{cell output}
\end{aligned}
\tag{3.10}
$$

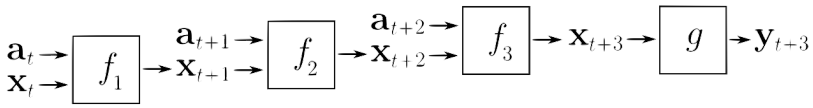
LSTM architecture has proven to be effective in language modeling. This thesis takes advantage of that and LSTM will be the predominant architecture used in the following experiments. However, one remaining essential part of the stack needs to be explained before diving into the implementation.

3.3 Training

To train recurrent neural network, same error function as with FFNNs can be used to calculate networks prediction loss 3.4. To backpropagate the error in time step t_n, we unfold the recurrent part of the network n times. In other words we use standard backpropagation 3.6, but we cannot treat state s_n as a constant because it depends on s_{n-1} and we need to apply chain rule to each state until s_0. Following image displays a simple recurrent neural network.



Unfolded neural network at time step $t = 2$ can be then visualized like this.



Now backpropagation algorithm from section 3.1.1.2 can be applied with a little tweak. We need to sum up the contributions of each previous time steps to the gradient because they all share weights W. Following equation demonstrates the calculation of the partial error of W for the time step $t = 2$.

$$
\frac{\partial E_2}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_2}{\partial y_2} \frac{\partial y_2}{\partial s_2} \frac{\partial s_2}{\partial s_k} \frac{\partial s_k}{\partial W} = \delta_2
\tag{3.11}
$$

This algorithm is called *backpropagation through time* (BPTT).

While vanilla RNNs have performed well on sequential data, more complex recurrent architectures tend to produce even better results.

3.4 Optimization

During the training of a neural network, the network may come to the state when it is overfitting. This means that the accuracy of network is increasing of the training data, but it is decreasing on the test data set. The network is losing the ability to generalize. There is a number of methods to prevent this. One of the methods is *early stopping* when the training process is programmed to detect increasing error on the testing data set. When detection is triggered, learning rate η in equation 3.5 can be decreased of the whole training can be stopped.

Another method to avoid overfitting is called *dropout*. During training each neuron has a certain probability p which describes how likely it is that the state of the neuron will be reset. This technique adds noise to the network and makes each neuron less dependent on the others and network is forced to use its nodes efficiently. Experiments have shown that dropout decreases error of predictions of neural networks [22].

3.5 Word embedding

It has been mentioned in the previous section that recurrent neural networks perform well in modeling sequential data, specifically language. The sequential time line can be represented as a sequence of incoming tokens. In language modeling, tokens can be symbols of the language such as the English alphabet with punctuation and white spaces. However, in the realm of language modeling we can bring individual tokens to high abstraction, representing whole words. The most basic method for word representation is indexing individual words in a vocabulary. Although this representation is easy to acquire and robust, it cannot capture any relationship between the words. This chapter discusses a neural modeling technique which allows us to store a word with its contextual meaning in a multidimensional vector. This approach has in some cases proven to outperform standard N-gram models with one hot word encoding [13].

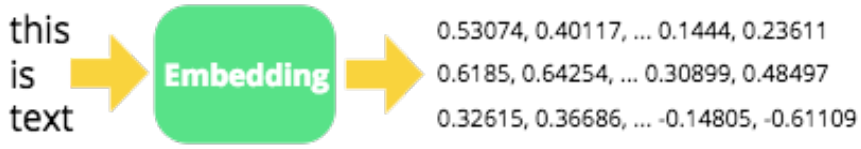


Figure 3.5: Transformation of words to embeddings

There are already existing tools to acquire such representations of words, for example the word2vec algorithm [15]. Input of the word2vec algorithm is a text corpus and output is a file with word vectors. The algorithm internally first constructs a vocabulary of the words and then using feed forward neural network learns their vector representation.

Word vectors hold some interesting properties. When algebraic operations are applied on word vectors, the resulting vector is surprisingly close to the result which we would intuitively expect.

$$vector(Paris) - vector(France) + vector(Italy) \doteq vector(Rome)$$

or

$$vector(King) - vector(Man) + vector(Woman) \doteq vector(Queen)$$

To observe the desired results, the model needs to be trained on a large data set using sufficiently large vector space. There exist a pretrained models such as Global Vectors for Word Representation [GloVe] model [19] or word2vec pretrained model [12]. Experiments in this thesis were conducted using mostly 300-dimensional and in special occasions 60-dimensional pretrained word vectors from pretrained GloVe model.

Chapter 4

Implementation

For implementation and training of language models, two different software stacks were considered.

4.1 Tensorflow

Tensorflow is an open source machine learning library developed by Google. It's core is written in C++ and it comes with Python wrappers. The advantages of Tensorflow are its strong support for distributed computing across many machines and also a vibrant online community. The main drawback is huge consumption of memory during training. This was a deliberate decision of Tensorflow's developers to accelerate the process of training, but for the purposes of this thesis it is inconvenient. Therefore, after a couple of experiments the alternative presented in the next section was chosen.

4.2 Keras and Theano

Keras and Theano are independent libraries, which together form a popular machine learning stack.

Theano is an open source library used for linear algebra calculations. It uses a NumPy-like syntax and is compiled to run efficiently on CPUs or GPUs.

Keras is an open source high-level neural network library. It runs on top of Theano, which it uses for fast computation. It offers a variety of preprogammed abstract structures such as FFNNs, RNNs, LSTMs and many more. All of these structures can be used directly and offer a variety of functions to adjust to adjust the architecture of the models. The process of training in Keras can be completely automated, but if needed Keras allows almost any training parameters to be manually adjusted. All these features make Keras very easy to use and in combination with Theano also reasonably fast during training.

4.3 Modeling

To reconstruct punctuation an appropriate language modeling needs to be chosen. The model needs to rely completely on the textual data of transcript. Contrary to [25], information about pause duration between words is not provided to the current model. Generally, the model is implemented as one or more RNN/LSTM blocks capped by one feed forward layer with a sigmoid activation function on its end. The first n layers of RNN/LSTM units are supposed to learn high level features from the textual data while the remaining feed forward layer is meant to combine these features into probability of a punctuation symbol after the current token. Two different ways of creating input for the network are proposed.

4.3.1 Input without context

When feeding the network with input *without context*, each token is represented by word vector of a single word. Therefore, when the network is computing probability of a punctuation symbol for token x_t, it holds in its memory information about all the previous tokens x_{t-1} ... x_0, but it has no information about the potential proceeding words which can come quite useful.

4.3.2 Input with context

On the other hand, when we want to provide context to the network, we need to construct tokens x_t differently. A token is represented by a matrix of N vertically concatenated word vectors. A punctuation symbol is modeled after M words, where $M < N$ and $M, N \in \mathbb{N}$. The remaining $N-M$ words are following after the hypothetical punctuation symbol. Order of words in the matrix is same as was given in the transcribed sequence of words. We can call this set of words a *word frame*. This way of creating input tokens gives network valuable information about the words after currently evaluated spot for punctuation symbol. Size of an input matrix is then (word vector size) $\times N$.

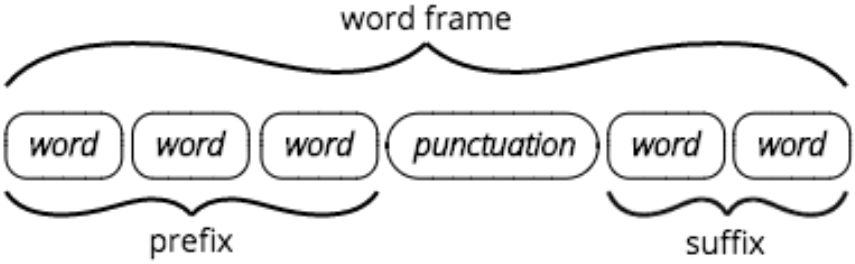


Figure 4.1: Sample word frame scheme

A problem arises with edge cases, when evaluating token x_t, where $t < M$ or
(total number of tokens $- t$) $< N - M$. These are the cases when we are evaluating punctuation symbol for a word too close to the beginning or the end of of a transcript. This can be easily solved by implementing special tag for „empty“ word. Empty words hold no semantic meaning and their sole purpose is to create a padding for word frames. Since empty words should hold no meaning, the word vector they are mapped to consists solely

of zeros. Because of this these words do not bring any noise to the system and also they do not accidentally trigger any neuron because all the weights are effectively factored to zero. Given this knowledge, each transcript would be prefixed by $M - 1$ empty word tags and suffixed by N empty word tags so that sufficient padding is provided for word frames.

4.3.3 Model parameters

Training error is computed as binary cross entropy. Models are trained using Adagrad optimizer, a modified version of gradient descent algorithm. Dropout with value 0.4 is applied on every RNN/LSTM layer. This decision was made because the experience with training recurrent models for punctuation reconstruction is that the models tend to overfit. Such a strong dropout partially eliminates this problem.

4.4 Processing pipeline

Most of the parts of the processing pipeline have already been mentioned, such as framework, model, etc., but it has not been described as whole yet.

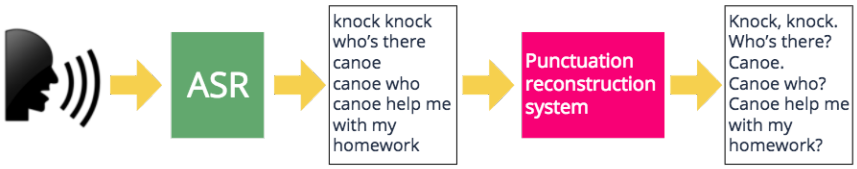


Figure 4.2: Scheme of processing pipeline

Figure 4.4 describes the whole process of transforming speech into well structured text. However, this section will discuss only the punctuation reconstruction system.

4.4.1 Preprocessing

One of the main reasons for preprocessing is to supply model with less noise in the data with goal of increasing its accuracy. There is a number of Python libraries which simplify this process, such as `nltk`, `pandas` or `scaPy`.

Loaded corpus first needs to be transformed into universal form, i.e. changing all characters into lower case or inserting white characters in appropriate places (surrounding punctuation symbols with white spaces so they would not be connected with proceeding words).

Special tags can give model extra metainformation about the text which could either reduce complexity or increase accuracy of classification. Special tags are either added by replacing other symbols (reducing complexity) or just added to specific places such as empty word tags discussed in section 4.3.2. When combined with word embeddings, they bring in the advantage of having unique, unused embedding. With multi-class classification, unique tags for each punctuation symbol are added. Tags need to have syntax different from the rest

of natural language to be easily parsed. In this project a XML-like syntax of tags was used, such as `</s>` as a replacement of period symbol of `</0>` for empty word. A sentence „Hello. I wish I had more time for this work. But I will do my best to finish it.“ would after processing look something like this „`</0> </0> </0> </0>` hello `</s>` i wish i had more time for this work `</s>` but i will do my best to finish it `</s> </0> </0>`“. This example is preprocessed for word window with prefix of four words and suffix of two words.

In some cases, preprocessing needs to be done as a prerequisite to create a corpus suited specifically for needs of some experiment. In this case a standalone script runs through the original corpora modifies them and creates various new corpora according to needs of experiments. In case of this thesis project, following approach was used to counter overfitting of model on one corpus. The goal was to develop a network which would be able to generalize on text from different corpora than the one it was trained on. The standard training corpus was Wall Street Journal (WSJ), however, performance on different corpora, e.g. CNN news transcript, was not so satisfying. The preprocessing script created four new corpora consisting of data from both, WSJ and CNN datasets. The ratio of WSJ vs. CNN data in new datasets was given according to logarithmic scale, i.e. 1/2, 1/4, 1/8 and 1/16. Sentences from datasets were extracted using `nltk` toolkit and shuffled into the new dataset. Results of this experiment will be discussed in the next chapter.

4.4.2 Data manipulation

Since datasets used in this project reach sizes of 3.5 GB, an efficient way of data manipulation needs to be used. If not careful, the training script can easily use up all the requested memory (when using Sun Grid Engine computational cluster) and even be terminated during training leading to loss of time and non-complete training data.

Fortunately, Python comes with a generator construct designed exactly for such a problems. A generator in Python can be created either as a function using key word `yield` or as a class. In both cases the construct acts as an iterable and therefore can be iterated over. The core principle of generators is that they are able to read, parse and modify data on the go. This feature significantly lowers memory requirements of the algorithm and makes generators very beneficial when working with large amounts of data.

For the purposes of this project two generators were used. A sequence generator for computing word frames 4.3.2 and corresponding targets (punctuation symbols) from labeled data. The next one is a batch generator which operated as a superset of sequence generator and used it to generate new batches of word frames and labels. These generator were used directly in training script to save memory.

4.4.3 Training script and classification

Training script can be considered as the core of the project. It uses preprocessing and data manipulation scripts, builds model, trains it on data and stores results.

For building models, Keras with Theano, and Tensorflow were used, each having its advantages and disadvantages already discussed in 4.1 and 4.2. Non of these options supported rapid model prototyping, though. Rapid model prototyping meaning creating new deep models parametrically, i.e. specifying architecture of the network with a parameter. For

this purpose a custom function on top of the Keras framework was created which later allowed for automatic testing of numerous different network architectures.

Keras in combination with Theano, which was the predominantly used framework, offered rich set of functions for training and allowed high control over the process while still working with high-level abstract functions. As mentioned in 4.4.2, the training process closely cooperated with generators to use memory optimally.

The standard way of storing trained model is saving the network architecture in a separate file encoded in JSON format. The weights are encoded in HDF5 format and also stored in separate file. This allows for later use of the trained network. It can be trained on further data of used for classification. Tensorflow lets you even create an executable program for classification from trained network. This is very useful in production and lets you circumvent loading of large Keras & Theano libraries and gives you almost instant time of classification.

Tensorflow also comes with a useful feature called TensorBoard. It a training visualization tool which gives you statistics and training progress graphs in well organized dashboard and is also able to send you notifications via the Internet on specified events during training. TensorBoard is, however, not compatible with Keras and Theano framework. To be able to use it, training output files need to be ported to required format and just then displayed using TensorBoard. This approach displays training data statically after the training is done. A better approach is to use Keras as abstraction library and Tensorflow as low-level library for linear algebra. This allows you to use all the features of TensorBoard, but comes with the expense of huge memory consumption, as it is already mentioned in 4.1.

Chapter 5

Training and Experiments

As for all machine learning models, the key part of their success is training them on datasets which represent their use case as accurately as possible. Science has not yet come up with a definite heuristic of how to determine the right network architecture and the best training hyperparameters. Although, recently there have been some attempts to come up with a meta-learning algorithms which can adjust their hyperparameters according to the problem [1] [20], this thesis takes a more conventional approach. Instead, a conventional approach to training neural networks was chosen.

First, the hyperparameters are set to values which have been known to perform well on similar problems. A baseline model is selected giving accuracy which we want to increase. A baseline model can be either a naïve solution to the problem or a model developed by somebody else which we want to outperform.

The hard part of creating a well performing model is then to select the best neural network architecture and training hyperparameters. Since there is no rigorous heuristic for that, the best performance is achieved by executing numerous experiments with different neural network architectures and probing through the space of hyperparameters until the best performing are found. Many skilled machine learning engineers even develop a kind of intuition for this process.

The baseline model used for this project was a classifier developed by Karel Beneš for Speech@FIT research group. Accuracy of some models developed for this project has increased by 14% against the baseline model.

5.1 Evaluation

To compare performance of different models a method of measurement needs to be established. Since this project operates with binary or multi-class classification, classification attempts can be divided into four groups: true positive, false positive, true negative and false negative (miss). Using these variables a statistical measure F_1 score can be computed.

F_1 score is a statistical measure ranging from 0 to 1 with bigger number meaning more accurate classification. To compute F_1 score for binary classification a confusion matrix needs to be created first. Confusion matrix is basically just a matrix holding count of true

positive, false positive, true negative and false negative classifications. These scores are collected at the end of each epoch by validating model on a validation dataset.

The confusion matrix is then aggregated into the F_1 value of the model at its current stage of training. Function for F_1 score operates with two variables, *precision* and *recall*.

Precision is defined as number of correct positive results divided by the number of all positive results. In other words it expresses how sensitive the model is to positive classification.

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{5.1}$$

Recall is defined as the number of correct positive results divided by the number of positive results that should have been returned. This value can be interpreted as ability of the model to stay conservative in some decisions and not to prematurely classify positively.

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{miss}} \tag{5.2}$$

The value of F_1 is then given by following equation.

$$F_1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} \tag{5.3}$$

Another important issue to consider when comparing and measuring accuracy is the dataset on which the models are trained and validated. To achieve the most objective comparison, the datasets need to be the same. Exception to this are experiments in section 5.6 where two different datasets were merged together with goal of avoiding overfitting.

5.2 Dataset

Datasets are often considered to be the key component of well performing machine learning model. Also, ideal datasets for a task are often hard to get by and acquiring such a dataset can be a task of its own. When creating a machine learning algorithm, it is helpful to have an idea of the data it will be processing when applied into business.

Punctuation reconstruction algorithm developed in this project is meant to be applied mostly on transcripts of lectures and public events. This means that a combination of prepared and improvised, formal and informal language is going to be processed. With this kind of language, especially informal, it is hard decide the correct punctuation and often many plausible solutions are available.

To simulate this kind of language, Wall Street Journal corpus and its subset, Pen Tree bank data set were used. Wall Street Journal corpus was created as a manual transcription of AP News collected over the period of years from 1988 to 1990 [27].

To measure accuracy of a model objectively and avoid overfitting a separate validation dataset is needed. Best case is when the validation dataset is from a different source than the training dataset. However, this is not always possible. Common practice is to split the main dataset into training and validation parts before training.

It should be also noted that experiments in section 5.6 are aimed to extend generalization of the network and the CNN news transcript corpus is used in combination with the Wall Street Journal corpus.

Both, the Wall Street Journal dataset and the CNN dataset were provided by my supervisor.

5.3 Wordframe experiment

Motivation behind this experiment was to decide the best feature selection; one-by-one 4.3.1 or wordframes 4.3.2. Conventional approach to using recurrent neural network is to predict next element according to the previous elements. However, the premise behind this experiment was that giving the network information about words succeeding the hypothetical punctuation symbol. This idea proofed to be correct.

Graph below shows progress of F_1 score calculated on validation dataset. A neural network with shallow LSTM architecture was used and they were trained with the same hyperparameters. The training dataset used for this experiment was Penn Tree bank corpus. Individual experiments differ in the number of words in contained in their suffix. Length of wordframe was 19 words and therefore length of prefix was 19 − len(suffix).

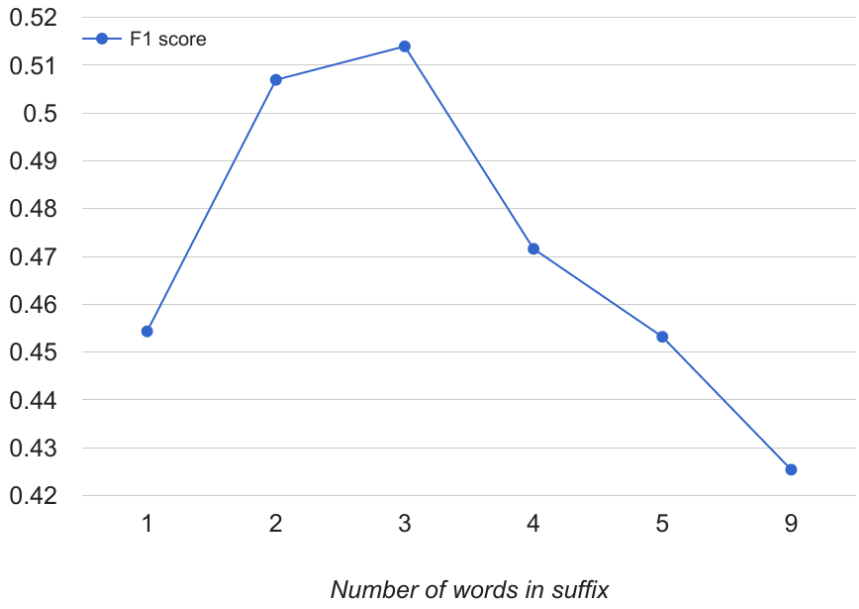


Figure 5.1: Comparison of F1 score for wordframe experiment

From the graph we can conclude that wordframes with suffix of length 2 or 3 have significantly higher accuracy then the rest. An LSTM network with input wordframe consisting of 5 prefixes and 3 suffixes was then trained on the whole Wall Street Journal dataset and reached F_1 score of **0.5824**.

Also other experiments have shown that length of prefix bigger than 10 makes insignificant difference in the final results and only increases complexity of the model.

5.4 Deep LSTM network experiment

Deep neural networks have gained great popularity due to their ability to recognize abstract features in complex data such as images [7] or voice [2]. Human language, especially spontaneous human language, is sequential data generated by complex algorithm with many abstract features. By this description human language seems as a perfect fit for deep neural network and its ability to recognize abstract features might come handy with punctuation reconstruction.

A number of tests were prepared to discover the number of hidden layers leading to the best classification accuracy. An LSTM neural network with wordframe input of 7/4 prefix/suffix ratio were used. All the experiments were executed using the same training hyperparameters. The models were trained on Penn Tree bank corpus. Individual experiments differ in number of hidden layers in the model.

Following graph shows change in F_1 score during the process of training. Deep LSTM networks with two, three, five and eight number of layers were trained and examined.

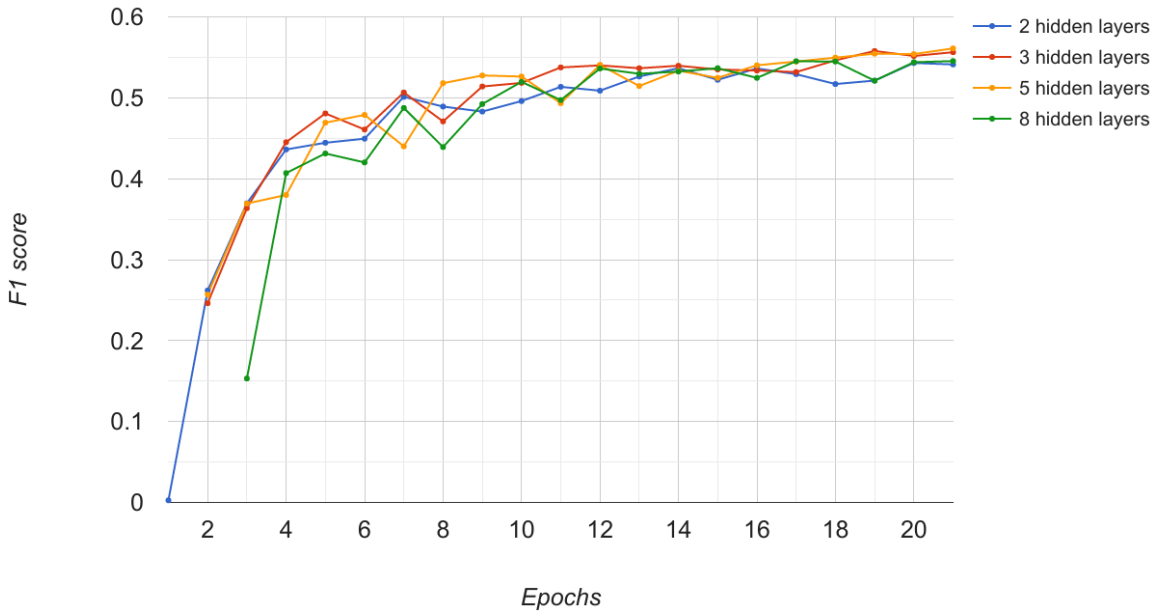


Figure 5.2: Training of deep LSTMs

From these experiments we can conclude that different number of hidden layers does not influence accuracy, but only increases the complexity of the model. We can also see that models seem to saturate from thirteenth epoch at value of approximately 0.53.

Following table shows final scores of the examined models after the last epoch of training.

# hidden layers	F_1 score
2	0.5436
3	0.5360
5	**0.5609**
8	0.5429

Table 5.1: Deep LSTM final results

Deep LSTM neural network with five hidden layers from the experiments above was also trained on the whole Wall Street Journal corpus. Resulting F_1 score was **0.6374**. The model was also trained on CNN corpus resulting into **0.7455** F_1 score.

The set of experiments on deep LSTM networks definitely proofed that deep models have significantly higher accuracy than basic shallow models. The deep models achieved higher accuracy when trained on larger datasets such as CNN corpus.

5.5 Deep vanilla recurrent neural network experiment

Recurrent neural networks have shown great potential [6] in natural language processing. Even though it is a significantly simpler model than long short-term memory architecture, it might still perform well on punctuation classification. Following set of experiments was set up to discover effectiveness of RNNs on this task.

A set of three deep RNNs was tested on Penn Tree bank dataset. RNNs with wordframe input of 7/4 prefix/suffix ratio were used. All the experiments were executed using the same training hyperparameters. Individual experiments differ in the number of hidden layers.

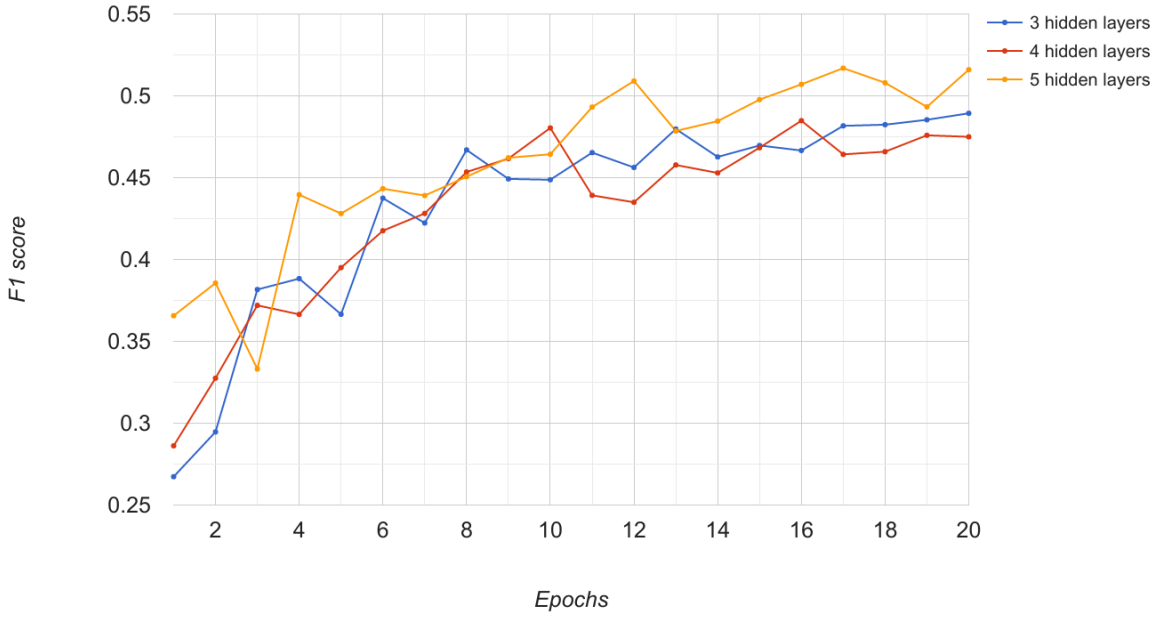


Figure 5.3: Training of deep RNNs

RNNs with three and four hidden layers seem to saturate at the tenth epoch. However, RNN with five hidden layers still tends to rise. Also we can see a lot more turbulence in F_1 score progression in the graph above than in the graph of LSTM training. We can conclude that deep RNN networks reach worse accuracy than LSTM networks with the best F_1 score **0.5246**. Since performance of these models did not surpass LSTMs, no further experiments with larger corpora were prepared.

# hidden layers	F_1 score
3	0.4872
4	0.4713
5	**0.5246**

Table 5.2: Deep RNN final results

5.6 Extending corpus experiment

Even though deep LSTM networks in section 5.4 were reaching great accuracies, their performance declined drastically when they were evaluated on CNN dataset (keeping in mind they were trained on Wall Street Journal dataset). Since production data is more general and broad then data in Wall Street Journal corpus, this issue needed to be overcome.

One suggested solution was to combine the two corpora and train a model on the newly created corpus. The combination was not trivial since the text needed to be treated not as a sequence of characters, nor a sequence of words, but a sequence of sentences where each sentence needed to keep its atomicity. Also special cases like dot symbol used after abbreviations needed to be taken in account. Although these special cases were relatively rare, `nltk` library came very helpful with its function for parsing sentences out of text.

A set of four new corpora was created with each corpus containing different proportion of Wall Street Journal and CNN corpora. The proportions were established logarithmically with first new corpus having a half of sentences from Wall Street Journal and a half from CNN. The remaining ratios were 1/4, 1/8 and 1/16 of CNN data in the new corpus. Legend on the left side of figure 5.6 symbolizes these portions of CNN sentences included in new dataset. The chart itself shows training progress of four models with same architecture and configuration of hyperparameters trained on these four different datasets. The evaluation was done on validation dataset which is one tenth of the training dataset.

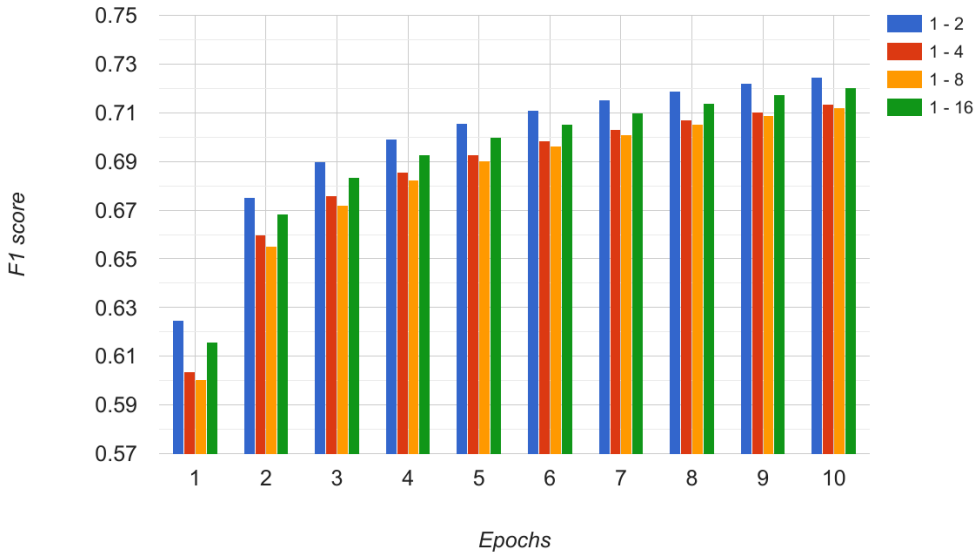


Figure 5.4: Training progress of corpus expansion experiment

Interesting properties can be deduced from these trainings. For example the network trained on a corpus consisting of half CNN half WSJ sentences had significantly better scores on validation dataset than the remaining networks. Interestingly, the network trained on a dataset consisting of only one sixteenth of CNN sentences performed also quite well on its validation dataset.

All of the models reached exceptionally high F_1 score of approximately 0.72 during validation. These results need to be taken with a grain of slat because as it is repeatedly mentioned, the evaluation was done on a portion of the training dataset. Following chart, however, displays evaluations done on AMI corpus. AMI corpus is a multi-modal data set consisting of 100 hours of meeting recordings. The nature of this corpus is more conversational than textual and therefore it models automatic speech recognition system outputs more accurately.

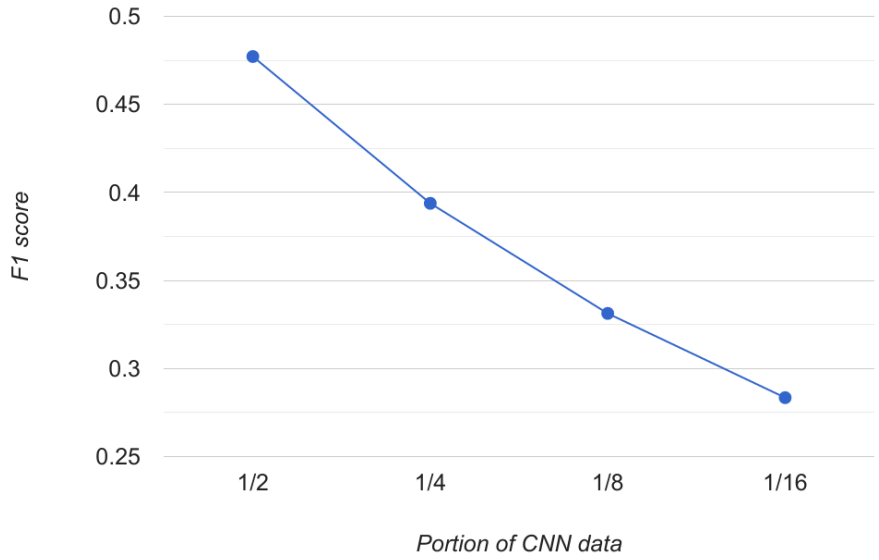


Figure 5.5: Test evaluation of expanding corpus models on AMI dataset

Although scores acquired from this evaluation have not reached any extraordinary values, a clear patter was discovered. The more CNN sentences (or less WSJ sentences) were included in the training dataset, the better results were achieved on AMI dataset evaluation. From this relationship we might imply that the more sentences from CNN corpus are included in the training dataset, the more accurate will be results of the model be when fed with actual speech transcripts.

Low F_1 score can be compensated by proportionally inflating the training corpus, i.e. keeping the ratio of WSJ vs. CNN sentences in the training corpus same, but increasing its volume. As it was already mentioned, sizes of the training corpora used in this experiment were 200 MB each. Since the size of the whole CNN corpus is approximately 3,5 GB there is definitely space for improvements. Following table displays exact F_1 scores achieved during evaluation by each model where the models are differentiated by proportion of CNN data in their training dataset.

proportion of CNN training data	**F_1 score**
1/2	**0.4771**
1/4	0.3938
1/8	0.3312
1/16	0.2834

Table 5.3: Test evaluation of expanding corpus models on AMI dataset results

Chapter 6

Conclusion

In this thesis, I have developed a punctuation reconstruction system for speech transcripts. I was particularly focused on language modeling using recurrent neural networks which I used for predicting punctuation. The predominantly used neural network architecture were long short-term memory networks.

The models were evaluated using Penn Tree bank, Wall Street Journal and CNN data sets. Evaluations have shown that networks fed with contextual tokens with specific ratio of prefix/suffix words achieve significantly better accuracy on predictions. Also, it was observer that deep long short-term memory networks have higher accuracy of predictions than shallow networks. Models with the highest accuracy reached F_1 score of **0.6374** and **0.7455** which is more than the baseline model created before at Speech@FIT and in this paper [25] by Ottokar Tilk. A pattern was discovered indicating that CNN corpus suits speech transcription data more accurately and models trained on this dataset achieve higher accuracy. I have also found several ways of how to execute trained models instantly which will be very useful in commercial applications.

Bibliography

[1] Abraham, A.: Meta learning evolutionary artificial neural networks. *Neurocomputing*. vol. 56. 2004: pp. 1–38.

[2] Bahdanau, D.; Chorowski, J.; Serdyuk, D.; et al.: End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*. IEEE. 2016. pp. 4945–4949.

[3] Elman, J. L.: Finding structure in time. *Cognitive science*. vol. 14, no. 2. 1990: pp. 179–211.

[4] Greff, K.; Srivastava, R. K.; Koutník, J.; et al.: LSTM: A search space odyssey. *arXiv preprint arXiv:1503.04069*. 2015.

[5] Hinton, G. E.; Srivastava, N.; Krizhevsky, A.; et al.: Improving neural networks by preventing co-adaptation of feature detectors. 2012. arXiv:1207.0580.

[6] Karpathy, A.: The Unreasonable Effectiveness of Recurrent Neural Networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/.

[7] Krizhevsky, A.; Sutskever, I.; Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*. 2012. pp. 1097–1105.

[8] Lu, W.; Ng, H. T.: Better punctuation prediction with dynamic conditional random fields. In *Proceedings of the 2010 conference on empirical methods in natural language processing*. Association for Computational Linguistics. 2010. pp. 177–186.

[9] Matusov, E.; Mauser, A.; Ney, H.: Automatic sentence segmentation and punctuation prediction for spoken language translation. In *IWSLT*. Citeseer. 2006. pp. 158–165.

[10] McCulloch, W. S.; Pitts, W.: A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*. vol. 5, no. 4. 1943: pp. 115–133.

[11] Miikkulainen, R.; Dyer, M. G.: Natural language processing with modular PDP networks and distributed lexicon. *Cognitive Science*. vol. 15, no. 3. 1991: pp. 343–399.

[12] Mikolov, T.: word2vec. https://code.google.com/archive/p/word2vec/.

[13] Mikolov, T.; Deoras, A.; Kombrink, S.; et al.: Empirical Evaluation and Combination of Advanced Language Modeling Techniques. In *INTERSPEECH*. s 1. 2011. pp. 605–608.

[14] Mikolov, T.; Karafiát, M.; Burget, L.; et al.: Recurrent neural network based language model. In *Interspeech*, vol. 2. 2010. page 3.

[15] Mikolov, T.; Sutskever, I.; Chen, K.; et al.: Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 2013. pp. 3111–3119.

[16] Minsky, M.; Papert, S.: Perceptrons: An Introduction to Computational Geometry (expanded edn). 1988.

[17] Olah, C.: Understanding LSTM Networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

[18] Peitz, S.; Freitag, M.; Mauser, A.; et al.: Modeling punctuation prediction as machine translation. In *IWSLT*. 2011. pp. 238–245.

[19] Pennington, J.; Socher, R.; Manning, C. D.: Glove: Global Vectors for Word Representation. In *EMNLP*, vol. 14. 2014. pp. 1532–43.

[20] Perez, C. E.: Taxonomy of Methods for Deep Meta Learning. March 2017. [Online]. Retrieved from: https://medium.com/intuitionmachine/machines-that-search-for-deep-learning-architectures-c88ae0afb6c8

[21] Shieber, S. M.; Tao, X.: Comma restoration using constituency information. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*. Association for Computational Linguistics. 2003. pp. 142–148.

[22] Srivastava, N.; Hinton, G. E.; Krizhevsky, A.; et al.: Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*. vol. 15, no. 1. 2014: pp. 1929–1958.

[23] Sundermeyer, M.; Oparin, I.; Gauvain, J.-L.; et al.: Comparison of feedforward and recurrent neural network language models. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013. pp. 8430–8434.

[24] Sutskever, I.; Vinyals, O.; Le, Q. V.: Sequence to Sequence Learning with Neural Networks. 2014. arXiv:1409.3215.

[25] Tilk, O.; Alumäe, T.: LSTM for Punctuation Restoration in Speech Transcripts. In *Sixteenth Annual Conference of the International Speech Communication Association*. 2015.

[26] Wang, D.; Nyberg, E.: A Long Short-Term Memory Model for Answer Sentence Selection in Question Answering. http://www.aclweb.org/anthology/P15-2116.

[27] Zhang, Y.; Callan, J.; Minka, T.: Novelty and redundancy detection in adaptive filtering. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2002. pp. 81–88.

Appendix A

CD Content

- `/`
  - `data/` – sample datasets used for training
    - `cleaning-scripts/` – scripts (not all) used for text preprocessing
  - `modeling-scripts/` – scripts used for building and training neural network models
    - `generators/` – Python modul containing batch and sequence generators
    - `utils/` – utility scripts used for plotting and sentence extraction
  - `experiments/` – configuration scripts for SGE and experiment results
  - `CNN_experiments/` – experiment executed on CNN dataset
    - `small_cnn_lstm_5d_35e/` – LSTM trained on a subset of CNN dataset
    - `smallsmall_cnn_lstm_5d_35e/` – LSTM trained on a smaller subset of CNN dataset
  - `deep_lstm_experiments/` – a number of experiments with different LSTM architecture
  - `deep_rnn_experiments/` – a number of sample experiments with different RNN architecture
  - `expand_corpus_experiments/`
    - `experiments/` – experiments trained on corpora with different CNN/WSJ ratio
      - `1_2/` – one half of CNN data
      - `1_4/` – one quarter of CNN data
      - `1_8/` – one eight of CNN data
      - `1_16/` – one sixteenth of CNN data
  - `n_last_experiments/` – a set of experiments with different word frame prefix/suffix ratios