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

RECOGNITION OF MULTI-TALKER OVERLAPPING SPEECH USING NEURAL NETWORKS

ROZPOZNÁVÁNÍ ŘEČI PŘEKRÝVAJÍCÍCH SE ŘEČNÍKŮ POMOCÍ NEURONOVÝCH SÍTÍ

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR AUTOR PRÁCE	JAROMÍR HRADIL
SUPERVISOR VEDOUCÍ PRÁCE	Ing. KATEŘINA ŽMOLÍKOVÁ

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Bachelor's Thesis Specification

Student: **Hradil Jaromír**

Programme: Information Technology

Title: **Recognition of Multi-Talker Overlapping Speech Using Neural Networks**

Category: Speech and Natural Language Processing

Assignment:

1. Get acquainted with automatic speech recognition of overlapping speech with permutation invariant training.
2. Get acquainted with convolutional neural networks and attention mechanism.
3. Implement the method using appropriate toolkit (e.g. PyTorch).
4. Train and evaluate on a conventional dataset, compare with published results.
5. Evaluate the results and suggest ways to further improve them.

Recommended literature:

- Chang, Xuankai, Yanmin Qian, and Dong Yu. "Monaural Multi-Talker Speech Recognition with Attention Mechanism and Gated Convolutional Networks." *Interspeech*. 2018.

- dle doporučení vedoucího

Requirements for the first semester:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Žmolíková Kateřina, Ing.**

Head of Department: Černocký Jan, doc. Dr. Ing.

Beginning of work: November 1, 2019

Submission deadline: May 28, 2020

Approval date: November 5, 2019

Abstract

This work deals with the speech recognition of overlapping speakers using a neural network. It examines the problem of speech recognition from multiple speakers and the ways in which this problem is solved. Specifically, in addition to traditional components such as convolutional neural networks, LSTM, etc., it is also an application of special components: attention mechanism and gated convolution. And also the application of a technique called permutation invariant training. Part of this work is to apply these approaches to assigned training data, which consists of artificially created mixtures of two speakers reading articles from the Wall Street Journal. The next step was to train the respective architectures using the combinations of the elements mentioned above. The models in this work replace the acoustic model. There were two architectures using different types of attention mechanism and one without it. Experiments have shown that architectures using the attention mechanism in this type of task have not surpassed more traditional architecture by suffering from gated convolution. Nevertheless, they showed potential.

Abstrakt

Tato práce se zabývá rozpoznáváním řeči překrývajících se řečníků pomocí neuronové sítě. Zkoumá problém rozpoznávání řečí od vícero řečníků a způsoby, jimiž se tento daný problém řeší. Jedná se konkrétně o aplikaci kromě tradičních komponentů jako konvoluční neuronové sítě, LSTM atd. také speciálních komponentů: attention mechanismus a gated konvoluce. A dále také aplikace techniky zvanou permutation invariant training. Součástí této práce je aplikování těchto přístupů na přidělená trénovací data, která jsou tvořena uměle vytvořenými směsmi dvou řečníků předčítající články z Wall Street Journal. Dalším krokem bylo natrénování příslušných architektur používající kombinující prvky zmíněné nahoře. Modely v této práci nahrazují akustický model. Jednalo se o dvě architektury užívající různé typy attention mechanismu a o jednu bez něj. Experimenty ukázaly, že architektury užívající attention mechanismus v tomto typu úlohy nępřekonaly tradičnější architekturu s užitím gated konvolucí. Přesto ale ukázaly potenciál.

Keywords

speech recognition,neural networks,attention mechanism,overlapping speech

Klíčová slova

rozpoznávání řeči,neuronové sítě,attention mechanismus,překrývající se řeč

Reference

HRADIL, Jaromír. *Recognition of Multi-Talker Overlapping Speech Using Neural Networks.* Brno, 2020. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Kateřina Žmolíková

Rozšířený abstrakt

Díky technologickému pokroku učiněném v posledních letech, zvláště v oblasti umělé inteligence a strojového učení, zaznamenala technologie automatického rozpoznávání řeči značný pokrok. Stále však jsou oblasti, kde rozpoznávání řeči nečiní tak velké pokroky. Jedna z těchto oblastí je překrývající se řeč vícero řečníků.

V rámci této práce je rozebrán jeden z možných přístupů, jak tento problém řešit. Jedná se o techniku nazvanou permutation invariant training (PIT), navrženou pro řešení právě tohoto typu úloh. Tato technika se aplikuje v rámci trénování neuronových sítí. Tato práce popisuje aplikaci této techniky na neuronové síti, která v rámci rozpoznávání řeči nahrazuje akustický model a přímo klasifikuje jednotlivé rámce. Model v této práci kombinuje tradiční prvky jako je např. konvoluční neuronová síť, LSTM, dopředná neuronová síť apod. a novější prvky využívané právě v rámci tohoto a podobných typů úloh. Jedná se o tzv. gated konvoluce a attention mechanismus.

Společně s těmito jednotlivými prvky pak byl příslušný model neuronové sítě trénován, aby dokázal rozpoznat překrývající se řečníky z datasetu WSJ0, kde jednotliví řečníci předčítají články z Wall Street Journal, a jejich směsi poté slouží jako základ pro onen model.

Příslušný model neuronové sítě byl implementován v jazyce Python s využitím knihovny pro strojové učení PyTorch. Jak již bylo zmíněno výše, neuronová síť na svém vstupu bere akustické příznaky, které posléze klasifikuje do příslušných tříd. Implementovaný model neuronové sítě nevyužívá attention mechanismus, který autoři navrhli v původním článku [3]. Místo toho byl vybrán jiný druh, který je méně náročný na výpočetní prostředky, navíc ve dvou variantách. V rámci experimentování byly trénovány celkem tři variace modelu s různou learning rate.

Experimenty s těmito modely ukázaly velmi zajímavé výsledky. V rámci trénování byly trénovány dva modely s attention mechanismem a jeden bez, který místo toho disponoval jednou vrstvou obousměrné LSTM navíc. Modely byly trénovány na jiném datasetu, než který byl použit pro evaluaci. Dataset pro evaluaci neobsahoval stejné řečníky jako tréninkový dataset. Hlavní metrikou pro určení úspěšnosti modelu byla široce užívaná metrika v rámci rozpoznávání řeči, WER (Word error rate). V rámci experimentů bylo zjištěno, že všechny modely mají tendenci se v pozdějších fázích trénování přetrénovávat (angl. overfitting).

Mimo to ale ještě předtím bylo také zjištěno, že příslušné modely kromě jednoho produkují překvapivé výsledky. Jedná právě o jeden z attention mechanismů, tzv. Multihead attention, který si v rámci experientů vedl nejhůře, jinak řečeno, neprodukoval tak dobré výsledky jako dvě zbylé varianty. Ty právě naopak vyprodukovaly velice dobré výsledky.

Nad všechna očekávání se jedná právě o model bez attention mechanismu, který vyprodukoval ze všech modelů nejlepší výsledky. Nejlepší výsledek tohoto modelu v rámci experimentů činí 28.4% WER, což je o 2.6% lepší, než nejlepší varianta modelu z původního článku, který dosáhl hodnoty 31%. I nejlepší varianta s tzv. Scaled Dot-Product attention mechanismem překonala nejlepší výsledek původního článku s celkovými 29.8% WER, tedy o 1.2% lepší. Nejlepší Multihead attention varianta naopak zaostávala s celkovými 34.5% WER.

Tyto experimenty ukázaly, že v rámci této úlohy mohou tradiční prvky společně s Gated konvolucemi a se správnou technikou dosáhnout lepších výsledků než modely s attention mechanismem. Ačkoliv attention mechanismus by mohl vykazovat lepší výsledky s vetším modelem, jelikož všechny modely obsahovaly z důvodu výpočetní náročnosti menší komponenty o jednu vrstvu BLSTM méně, než model bez attention mechanismu.

Modely by mohly dosáhnout ještě lepšího výsledku při aplikaci tvz. regularizačních technik, které se právě používají jako prevence jevu přetrénování.

Recognition of Multi-Talker Overlapping Speech Using Neural Networks

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Ing. Kateřina Žmolíková.

I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

. . . .
Jaromír Hradil
May 27, 2020

Acknowledgements

I would like to express my eternal gratitude to my supervisor Ing. Kateřina Žmolíková for her great support, inestimable advices, patience and time she dedicated me during consultations and mainly during experimental stage of the work.

Contents

Chapter 1

Introduction

As modern deep learning technology research progresses, there are more and more requirements from it, including a field of speech recognition. Today's level of speech recognition is very high when compared to the past decade yet there are still various cases, where the advancement of speech recognition is not so great. One of these cases in this field is an overlapped speech from multiple speakers at the same time. While speech recognition of one person is well-developed, multi-talker speech recognition is not. It is due to its greater complexity when compared to the speech recognition of one person.

However, with mentioned technological advances there are developed various approaches on how to resolve the problem of multi-talker speech recognition. One of the possible strategies was developed by researchers in [3]. They propose the use of a neural network with a new architecture containing new elements, namely gated convolutional layers and attention mechanism. This type of architecture along with a training technique called permutation invariant training should be able to improve the level of the multi-speaker speech recognition. The main goal of this thesis is to implement a neural network architecture using these new elements with the use of the machine learning library and to evaluate it on a standard dataset.

The thesis is structured into various chapters. Chapter 2 introduces the concept of neural networks and their use in speech recognition. Chapter 3 describes the principles and basic methods of speech recognition and multi-talker scenario. Chapter 4 describes the nature and characteristics of a dataset used for the training and evaluation of the network. Chapter 5 contains information about the implementation of the network based on the mentioned article [3] and chapter 6 summarizes experiments realized with implemented network.

Chapter 2

Neural networks

This chapter describes a concept of the neural network, the type of neural networks that are used in *Automatic speech recognition(ASR)*, and the other elements used to improve their overall performance. Information in this chapter about a structure of the neural network and its training mechanics were derived from publications [2, 7, 17].

2.1 Inner structure

The concept of the neural network started in the 1950s and 1960s when scientists discovered an artificial neuron. They called it *perceptron*. This unit is connected to several inputs. These inputs can be equal only to one or zero. Perceptron then takes these inputs and multiplies them with a set of parameters called *weights*. They represent the strength of the individual connections, i.e. how much these connections will influence the output. In the case of the perceptron, the output can only be equal to either zero when the weighted sum of its inputs is lesser or equal to zero or one when the weighted sum is greater than one which causes the perceptron to be activated. Today the artificial neurons are more sophisticated and can produce more outputs than only zero or one when other functions are used for their activation. These neurons can then be put together and influence each other. There are in total three different possibilities of how to connect neurons. This depends on their position in the model.

The first possibility is to use them as inputs of the model inside the *input layer*. Here they take input information and send it more forward into the model. The second possibility is to connect them in the part of the model which is referred to as *hidden layer*. In this layer neurons forward the inputs they receive further through the model. There can be multiple hidden layers. The number of the hidden layers then determines what „depth“ model has. The final possibility of how to connect neurons is to use them as the output units, which form the *output layer*. These principles described above are typical for one class of neural networks called a *feed-forward network* or *multilayered perceptron*. This type is typical for having all the connections only in one direction, i.e. it does not have any connections which would feed the output of any of the neurons back to the model. This is depicted in figure 2.1.

Neurons can use linear or non-linear activation functions. Linear functions have limited capabilities over those non-linear. Non-linear functions allow to perform non-linear transformations. Mathematically this process can be expressed for concrete neuron by the following formula:

$$y = a(\sum_{i=1}^{n} x_i w_i + w_0) \tag{2.1}$$

where a stands as the activation function, w_0 represents a bias value which is a value not connected to any input and which can influence the activation of the neuron. x is a vector of the inputs, w is a vector of the weights and n is a number of the input connections. Currently one of the most recommended activation function is called *ReLU* or *rectified linear unit*. Others quite popular are *logistic sigmoid* and *hyperbolic tangent*.

The output neurons also use activation function on their outputs depending on how they should be interpreted. For example, if outputs of the neurons are to be interpreted as probabilities they use a *softmax function*. To learn, the model must compute the gradients of the selected activation functions. The next section describes the characteristics of learning during the training process.

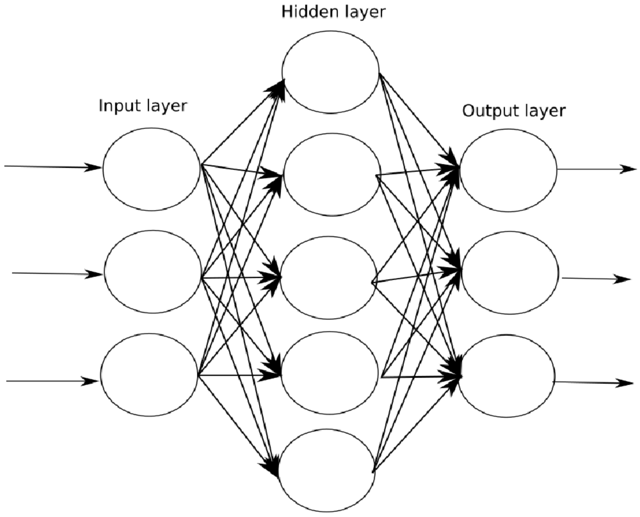

Figure 2.1: An example of a feed-forward network with one hidden layer, inspired by figure from [17].

2.2 Training process

Training is defined as a process, during which the model runs iteratively through the set of training data and modifies the weights of the neurons. These modifications allow the model to learn and to improve its performance after each iteration.

The whole process of training can be divided into three steps:

1. Forwarding of the data through the network and calculation of the *loss* value indicating how well model performs, calculated by a *cost function*.

2. *Backpropagation algorithm* for a calculation of the *gradient* of the cost function with respect to the weights of the neural network.

3. An update of the weights with a method called *gradient descent* which is an optimization algorithm used to minimize a *loss*. Gradient descent modifies weights in a direction of decreasing the value of the cost function.

There are various types of the cost functions used in neural network models, the most commonly used are *mean squared error function* and *cross-entropy function.* Loss L of the *cross-entropy function* for produced *n-th* output probability y_k^n and C target classes is defined by following formula:

$$L = -\sum_{n}\sum_{k=1}^{C} t_k^n \ln y_k^n \tag{2.2}$$

Where t_k^n is probability indicating that the target value equals to the class C.

Every model has various parameters called *hyperparameters* which are used to tune the model's performance. One of these hyperparameters is *learning rate.* This hyperparameter needs to be tuned correctly. It determines the size of a step used in the optimization algorithm when moving towards a minimum of the cost function. Otherwise, the training could diverge or its progress would be too slow.

2.2.1 Cross-validation and overfitting

To discover whether the model is capable of learning and recognizing new related data not present in the training dataset, a technique called *cross-validation* is performed. During this process, the model encounters a new type of data separated from the training dataset. Apart from that during this process, the model does not use the backpropagation algorithm.

This technique is very beneficial because it can indicate the behavior of the model through the calculated loss of the cross-validation dataset. Then it can be compared with the loss of the training dataset from the corresponding iteration and the comparison can reveal if the model is training the right way or not. Typically is very distant from the loss of training data. One of the typical problems is called *overfitting*, a situation when the model produces correct predictions on the training data but incorrect for data not present in the training dataset.

There exist some techniques to prevent this phenomenon. These techniques are commonly known as *regularization techniques.* There are various regularization techniques used among the field, here are some of the most used:

- **Dataset augmentation** - augmentation of the training dataset causes that the network will not overfit so easily due to the larger number of data. However, because obtaining more data for training can be expensive and difficult, there is a possibility to enlarge the dataset artificially. This means that it is possible to take training data and then modify them to be the same with a slightly different form. However, this approach is not fitted for every task, for example, a density estimation task.

- **Parameter norm penalties** - these techniques add a parameter norm penalty into the cost function. The penalty is chosen so that only the weights are penalized, not the biases. The reason behind it is that the biases require fewer data to fit and as a result, they do not add much of the variance. One of the most widely applied is L^2 *regularization* or *weight decay.* These techniques cause the model to learn smaller weights and the larger ones will be included only if their inclusion will reduce the loss of the cost function.

- **Dropout** [23] - this technique changes the model itself by temporarily dropping a few neurons from the hidden layers by multiplying their outputs by zero. The model calculates a random binary mask for each training sample. This mask then decides what particular neurons will participate in the training process. Because training with the Dropout technique will cause more of the hidden neurons to be active after training due to larger weights, the weights are scaled by $\frac{1}{p}$, where p is a probability that determines if the neuron is to be retained. This is called the *weight scaling inference rule.*

- **Early stopping** - this technique involves an observation of the validation loss. If the validation will get only worse with further training, the training is stopped and the best state of the model is stored.

2.3 Convolutional neural network

Another widely used type of neural network, besides the feed-forward network, is called *convolutional neural network (CNN).*

Neighboring neurons from the output layer are connected to other neighboring neurons from the input layer [16]. The whole operation can be understood as extracting features from the input data by performing a convolution operation with a set of learnable *kernels*, a feature extractor of a certain size moving all along the input data and scanning them [7]. The convolution can be performed along vectors or matrices [14]. The convolution of matrices which is also used in the implemented model is defined by the following formula [14]:

$$c_{xy} = \sum_{u}\sum_{v} a_{uv} b_{x-u+1,y-v+1} \tag{2.3}$$

where the convolution operation with matrix A of $n \times m$ size representing kernel is performed over the input matrix B of $k \times l$ size produces the output matrix C is of the $(n + k - 1) \times (m + l - 1)$ size, u and v represent ranges of all legitimate moves of convolution for elements a_{uv} and $b_{x-u+1,y-v+1}$.

Then in the next layer, called *subsampling layer*, the network will reduce the dimensionality of the produced output [16]. This operation, also referred to as *pooling*, helps to prevent output values from getting affected by small translations of the input [7]. This process is illustrated in figure 2.2. It is also worth to mention that neurons in the CNN layer share a single vector of weights among them [16].

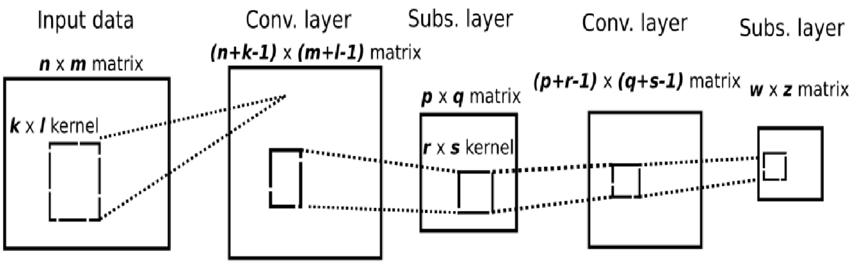

Figure 2.2: An example of convolutional neural network, Inspired by figure from [16]

2.3.1 Gated convolution

Details about the gated convolutional layer were derived from [3]. In the article [3] authors proposed the use of a special type of the convolutional layer, called gated convolutional layer displayed in figure 2.3. The layer consists of two separate convolutional layers each one having their weight parameters. One of the layers is using *sigmoid* activation function on its output, while the other uses linear activation. The outputs are then merged using element-wise multiplication thus producing the final output of the layer. This process is described by the following formula:

$$h(X) = (X * W + b) \otimes \sigma(X * V + d) \tag{2.4}$$

where X is the input matrix, W, V are weight parameters, b, d are biases, σ is the sigmoid function, $*$ is the convolution operation and $\otimes$ is the element-wise product. One of the capabilities of this layer is that it can control the data stream of its output.

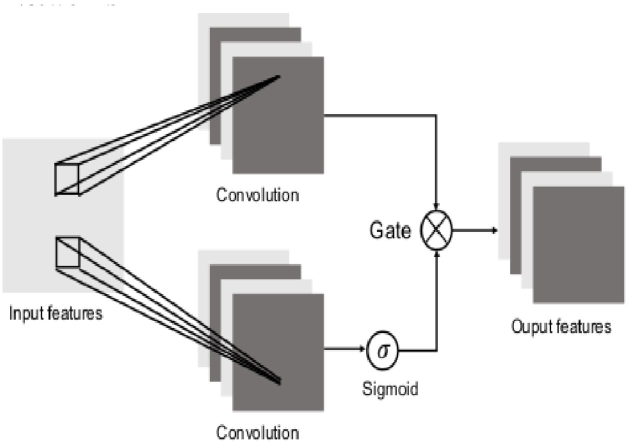

Figure 2.3: A Gated convolutional layer structure taken from [3]

2.4 Recurrent neural network

Details about the Recurrent neural networks were derived from [7, 18]. *Recurrent neural network (RNN)* is a type of neural network that is capable to use its previous outputs to produce new ones. It does so by sending the produced output through the connection back to itself. The RNN can be also seen as a chain-like structure, as depicted in figure 2.4. Here, block A corresponds to the computation performed in one time-step, with input x_t and output h_t. The outputs are reused in the next time step. Reusing these outputs helps RNN to learn the patterns between the elements of the input sequence.

RNNs are due to their characteristics widely used in tasks like speech recognition, language modeling, etc. However, RNNs have a problem in cases of *Long-Term Dependencies*, i.e., memorization of the previous inputs for a longer period of time.

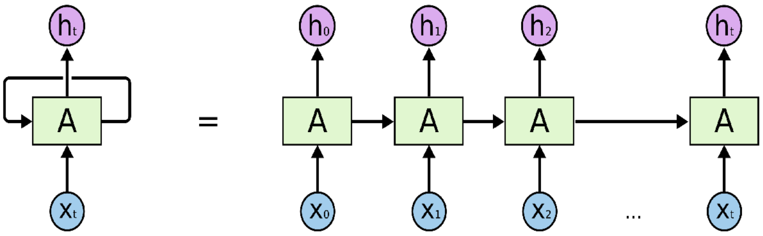

Figure 2.4: Aa example of the RNN structure schematic taken from [18]

2.4.1 Long short-term memory network

Details about Long Short-Term Memory (LSTM) were derived from [6, 18]. Long short-term memory (LSTM) is a subclass of RNN. It was created to counter the shortcoming of the RNN. Like the RNNs, LSTM also can be depicted as the chain-like structure as shown in figure 2.5. A calculation of a next state is realized in one time step by block A, but in case of the LSTM, the block is more complex, when compared to the RNN block in the figure 2.6.

In comparison with RNNs, LSTM additionally contains a cell state value C_t, a horizontal line running through the top of the figure 2.5. It is a value capable of storing information, which is transferred between particular time steps. It realizes this with the use of a component called a *memory block*, a structure containing a cell state. Then there are gating units, units with a sigmoid neural net layer, and pointwise multiplication operation, as can be seen in 2.5, which can let pass information through that particular block. These gating units are responsible for the protection and control of the cell state in the block. Cell state is in every step modified depending on the actual input and hidden state, with the use of forget gate (forgetting the information) and input gate (inserting the new information). The output is then calculated from the actual input, hidden state, and the actual cell state with the use of an output gate.

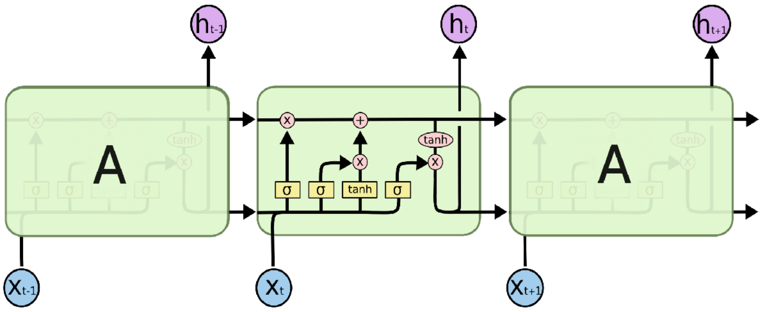

Figure 2.5: An example of the LSTM block structure taken from [18]

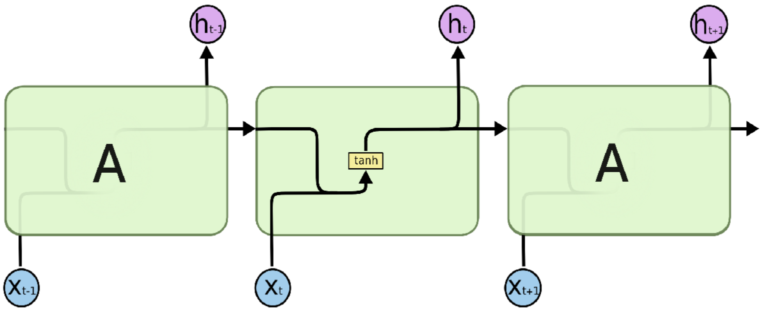

Figure 2.6: An example of the normal RNN block structure taken from [18]

2.4.2 Bidirectional LSTM

Apart from the standard RNNs, there also exists another type of RNNs architecture called *Bidirectional RNN* (BRNN). It is a type of RNN network, processing input sequence in two directions - forward and backward [7]. This allows them to capture not only the past but also the future context of the input information [7]. Into this group also belongs *Bidirectional LSTM* (BLSTM), originally introduced in [8]. BLSTM type is also used in the model of the proposed architecture from [3].

2.5 Attention mechanism

Attention mechanism is used in machine learning, especially in language translation, image captioning, dialog generation, etc. as a component capable of showing to the network where to concentrate its focus when making predictions of an element by his relations to the others. It does so by creating an attention vector, a vector carrying pieces of information about relations of the element to other elements in the form of the *attention weights* [25]. Description of the attention mechanism was derived from [24, 25]

Although a different type of *attention mechanism* is proposed in the original article [3], we use a different type in this work because of its computational efficiency. It is the *attention mechanism* based on the paper [24] which proved to produce *state-of-art* results. Its main advantage over the other types of mechanisms consists of the possibility to realize the process without the use of RNN, a common element among the other types.

This architecture uses linear transformations to produce three different vectors called *keys, queries*, and *values* from the input. Keys and queries have the same size of dimension d_k and values are of dimension d_v. Attention function then maps key-value pairs and queries to output as shown in figure 2.7. At the start, a vector of weights is calculated by a compatibility function of the keys and queries to assign a score in this type of the architecture defined by the following formula:

$$\mathbf{score}(Q, K) = softmax(\frac{QK^T}{\sqrt{d_k}}) \tag{2.5}$$

where K and Q are packed matrices of the *keys* and *queries* vectors. $\frac{1}{\sqrt{d_k}}$ represents a *scaling factor* where dimension d_k is an input sequence length. Its use is motivated by a possibility of the long input which could cause that softmax function will produce a very small *gradients* which are ineffective for learning. Then all weights are assigned to the corresponding values from packed matrix V and after that, a result is calculated as a weighted sum of the values, producing the Scaled Dot-Product attention:

$$\mathbf{Attention}(Q, K, V) = \mathbf{score}(Q, K)V \tag{2.6}$$

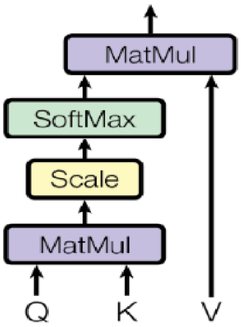

Figure 2.7: A Scaled-Dot attention mechanism principle, Figure modified from [24]

This principle is then developed further in the [24]. It proposes to use *Multihead Attention*. This principle proposes instead of using only a single attention function to use linear projections of the *queries, keys* and the *values* **h** times with different projections to the dimensions of the *keys* and *values*, shown in figure 2.8. This approach is defined by following formula:

$$\mathbf{MultiHead}(Q, K, V) = Concat(head_1, head_2, head_3....head_h)W^O \tag{2.7}$$

where the head is defined as:

$$head_i = \mathbf{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

where QW_i^Q, KW_i^K, VW_i^V, and W^O are projection matrices to be learned.

This principle allows calculating multiple outputs by different layers called *heads*, which are then concatenated into the final output. This allows the model to learn to attend with every single head to the different parts of the input information. This is according to [24] inhibited when using only a single *Scaled Dot-product* layer, by an averaging of the output.

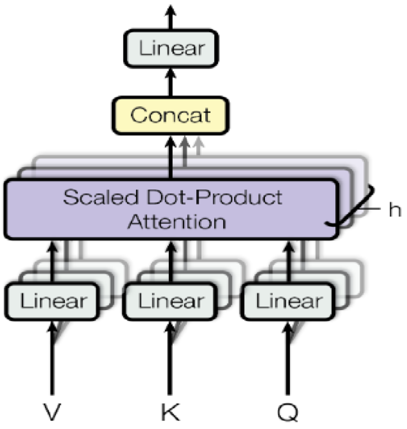

Figure 2.8: A multihead attention mechanism principle taken from [24]

Chapter 3

Multi-talker speech recognition

This chapter describes the functioning and architecture of the Automatic speech recognition (ASR) system. As next, it explains the basics of functioning of an acoustic model and its importance in ASR, methods that are used in the field of speech recognition, separation of speech, and an application of the acoustic model in the multi-talker scenario.

3.1 Overview of Automatic speech recognition system

The main sources of information about Automatic speech recognition (ASR) architecture were derived from [4, 22]. *Automatic speech recognition (ASR)* refers to a transformation of input speech waveform into a sequence of recognized words. In practice, there are various approaches to how to process the speech signal. One approach is called *Dynamic Time Warping*. Although this technique proved to be efficient in matching words, it does not perform so well in case of recognition of a continuous speech. Due to this fact, a different approach is used in the recognition of continuous speech.

Figure 3.1 demonstrates the ASR system, where a *decoder* aims to find a sequence $\hat{W}$ of words W which best matches X representing an acoustic feature sequence extracted from the input data. This can be expressed by the following formula:

$$\hat{W} = \arg\max_W \; P(W|X) \tag{3.1}$$

where

$$P(W|X) = \frac{P(X|W)P(W)}{P(X)} \tag{3.2}$$

In this formula $P(X|W)$, which is modeled by an *acoustic model* represents how well sequence X corresponds with a word sequence W. $P(W)$ represents the probability of the word sequence to appear in the current language. This probability is computed by a *language model*. $P(X)$ represents a prior probability of the feature sequence independent from the *acoustic* and *language model* and can be ignored. Valid words of the particular language are composed of the sequences of the basic unit of sound called phonemes. They are specified by a *pronunciation lexicon*.

3.2 Acoustic model

Details about the acoustic model were derived from publications [11, 22]. Before sending a speech signal into the *acoustic model*, the extraction of the features must be performed.

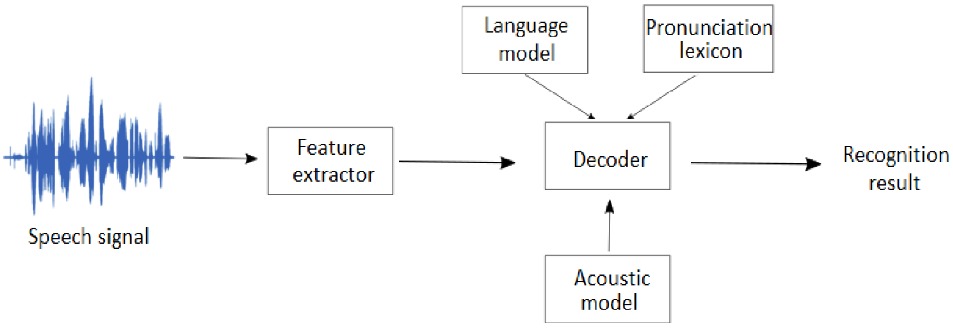

Figure 3.1: An architecture of the ASR system, inspired by figure from [4]

This transforms the signal into the form better fitted for speech recognition. The signal is divided into *frames* representing approximately 25 ms of time of the signal with time shift between successive overlapping frames being typically 10 ms. After that, a vector of features is extracted from each of these frames.

After the extraction, the features are sent into the *acoustic model.* In ASR, the acoustic model is a component that can greatly affect the overall accuracy. Upon the extracted features, the acoustic model creates their statistical representation. One of the most applied acoustic models is the Hidden Markov Model (HMM). HMMs are used due to their capability of representing the acoustic features and matching sequences of variable length.

HMM, can be viewed as Finite-State automaton. It is displayed in figure 3.2. This automaton has probabilistic transitions between individual states, i.e. their total sum must be equal to one. Let a_{ij} be a transition probability between *i-th* and *j-th* states. Following formula explains this relationship for N possible transitions from *i-th* state:

$$\sum_{j=1}^{N} a_{ij} = 1 \tag{3.3}$$

The model can transition into another state or stay in the current one. Transitions occur every time frame of time t. Each of the states, when entered, generates an observed acoustic feature vector.

HMM also contains two special states. Both are reached only once when the model starts and terminates its task. Apart from these two every state as mentioned generates an observation vector x_t through the emitting probability distribution $b_j(x_t)$. Using these elements it is possible to formulate a calculation of the likelihood of acoustic feature vector X given λ which represents acoustic components representing concrete words from sequence W by the following formula:

$$P(X|\lambda) = \sum_{s} P(s, X|\lambda) \tag{3.4}$$

Here s represents a vector of the states of HMM.

The joint probability of vector X and states in s given λ is formulated this way:

$$P(s, X|\lambda) = \prod_{t=1}^{T} b_{s_t}(x_t) a_{s_t s_{t+1}} \tag{3.5}$$

Here s_{t+1} represents an exit state of the model.

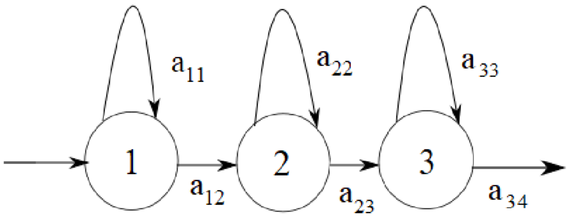

Figure 3.2: Model of phoneme states using HMMs, inspired by figure from [11]

3.2.1 Neural networks in acoustic model

The main sources of information for this subsection were derived from the [10].

As mentioned in the previous section, HMMs are a very popular tool in ASR systems. HMMs began to be used widely in this field with the discovery of the Expectation-Maximization algorithm. This method is used for training HMMs. This algorithm allowed to represent the probability density for each HMM state through *Gaussian mixture models* [10]. „*A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities.*“ [21]. GMMs and HMMs were since then used in combination as GMM-HMMs.

However, GMMs aren't very successful when the data require a non-linearity for their representation. Because of this, the neural networks began to be used instead. The neural networks are used because of their capability of modeling complex systems requiring a non-linearity and they showed to outperform GMM-HMMs.

In current time, neural networks as the acoustic model consist of many hidden layers and produce large outputs where each output corresponds with a certain state of the HMM.

3.3 Multi-talker acoustic model

3.3.1 Speech separation

As was mentioned in the introduction chapter, despite great advances in speech recognition tasks there are still scenarios really difficult to handle. One of them is the ability to recognize the speech from the noisy environment along with other people talking simultaneously. This problem is often addressed as a *cocktail-party* problem [26]. It tends to get even worse in the case of a single-channel speech mixture [12]. This is also the case when there is no prior information about speakers themselves, a *speaker-independent* scenario [12]. Here come monoaural speech separation techniques, whose goal is to extract the individual source of each speaker from the mixture.

One of the approaches how to deal with the multi-talker scenario is to separate individual signals from the mixture and then send those separated signals as the input into a classic single-speaker acoustic model.

For the separation process, there are two most popular approaches. One of them is called *Deep Clustering* [9]. „*Deep clustering is a recently introduced deep learning architecture* *that uses discriminatively trained embeddings as the basis for clustering.*“[12]. However, this method has its shortcomings because of the clustering step, it takes as a prerequisite that the time-frequency bin is dominated by only one speaker and although it proved to produce good final approximations, it is not an optimal way [26].

The second approach is called *Permutation invariant training* (PIT) [26] Using this approach, as can be seen in figure 3.3, at first, the neural network model produces the output masks from the multiple output layers and then these masks are used to produce individual signals [26]. Then using the *cost function*, the total losses of separated signals and targets are computed, however, because there is no prior information about speakers, there is no possibility to determine to which separated signal corresponds to which targets, a situation known as a label permutation problem [26]. To counter this situation PIT approach calculates loss for each permutation between all predictions and targets, then summing those obtained losses to produce a total loss of that permutation [26]. Then total losses of those permutations are compared and the least loss value is selected to optimize the model [26].

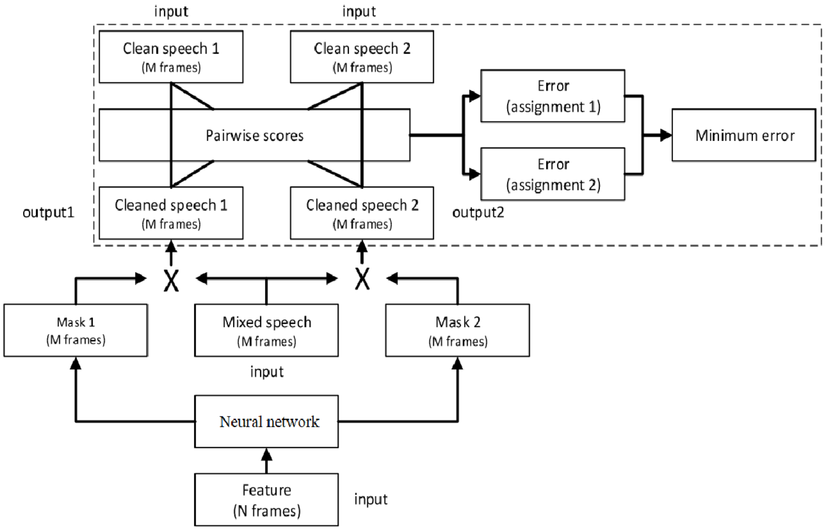

Figure 3.3: Process of permutation invariant training of 2 speakers, Figure modified from [26]

3.3.2 Direct multi-talker acoustic model

Instead of realizing the process of separation separately from the acoustic model, there is a possibility to design the acoustic model in a way, that it can recognize the speech of two speakers. In [3] authors use the neural network with the PIT cost function, however, this time it does not estimate the separated signal, it directly classifies the individual frames into phonemes. This whole process for S speaker sources can be formulated this way [3]:

$$J = \frac{1}{S} \min_{s' \in permu(S)} \sum_{s} \sum_{t} CE(l_t^{s'}, O_t^s), s = 1, ..., S \tag{3.6}$$

where $permu(S)$ is the set of permutations of the 1...S, CE is *cross-entropy* function, $O^S, s = 1, ..., S$ are output predictions produced by the model and $l_t^{s'}$ are corresponding targets.

Chapter 4

Dataset

This chapter describes the main characteristics of a speech dataset used for training and evaluation of the implemented models.

4.1 CSR-I (WSJ0)

CSR-I (WSJ0) [5] is the name of a corpus that was used for the creation of another dataset, originally introduced used by MERL in the paper [9]. The complete corpus also known as *LDC93S6A* consist of two parts: *LDC93S6B* and *LDC93S6C* [5]. *LDC93S6B* part contains utterances captured by a close-talking Sennheiser HMD414 microphone and *LDC93S6C* part contains utterances captured by a secondary microphone [5]. Both parts mainly consist of recorded utterances of persons reading Wall Street Journal articles and texts selected for reading for both parts had to satisfy a requirement to fall within a 5,000-word or a 20,000-word subset of the Wall Street Journal text corpus [5].

These recordings were then artificially mixed under the name „**wsj-2mix**“ by MERL in [9]. This mixtures form in total three datasets: training, cross-validation and test dataset of various lengths shown in table 4.1:

Dataset type	Length in hours
Train	30
Cross-validation	10
Test	5

Table 4.1: Table of dataset lengths

The process of creation of these datasets is the same is practically the same for all of them [9]. It consists of selecting randomly individual utterances of different speakers from the particular dataset and then these utterances are paired and mixed at various random *signal-to-noise ratio* in a range between 0dB and 10dB [9]. For the creation of the first two datasets was used the WSJ0 *si_tr_s* dataset [9]. For the creation of the test evaluation dataset were used the WSJ0 development dataset *si_dt_05* and evaluation dataset *si_et_05* where the utterances were selected from 16 different speakers which are not present in other datasets [9].

4.2 Test evaluation dataset

The test dataset used for the evaluation of models during the experiment stage contains different speakers from those in training and cross-validation datasets as mentioned in the previous section. There are in total three different groups of overlapped utterances: 1. mixtures containing only male speakers, 2. mixtures containing only female speakers and 3. mixtures where one speaker is male and the other one is female. The total share of each of the groups can be seen in the following table:

Gender	Number of utterances
Men	867
Women	530
Mixed	1603

Table 4.2: A distribution of utterances from test dataset by gender of speakers

During the experiments with test validation set the whole utterances were kept in the original state and processed individually, i.e. they weren't segmented by cutting procedure explained in subsection 5.2.2. It was because otherwise the WER metric couldn't be applied as it measures whole words and in addition, there would not be otherwise possible to observe if models did learn continuity patterns in the utterances. The following figure 4.1 shows the length of individual utterances used in experiments.

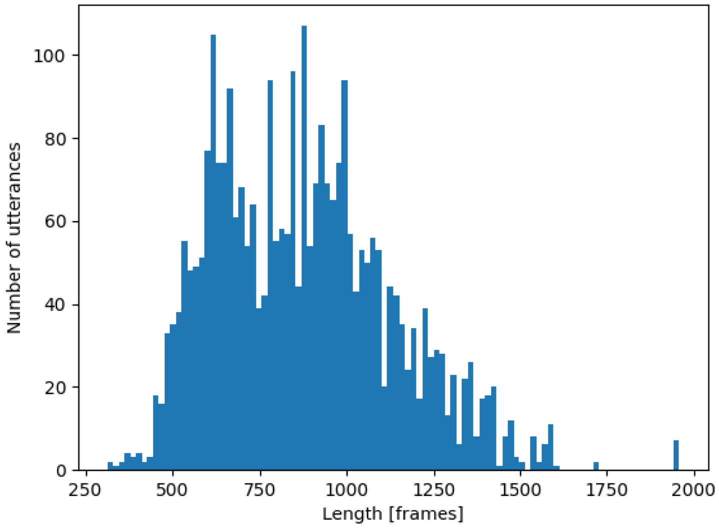

Figure 4.1: Histogram of lengths of utterances of test dataset.

As mentioned in section 4, the mixtures were created by mixing utterances at various SNR. Figure 4.2 shows an overall distribution of utterances regarding the values of SNR of the first speaker over the second.

Signal-to-noise ratio (SNR) [13] is the measure which is used to determine how is the signal powerful over the background noise. It is measured in decibels (dB). It calculated by the following formula:

$$SNR = 10 \log_{10} \frac{P_S}{P_N} \tag{4.1}$$

where P_S is the power of the signal and P_N is the power of the noise.

It is possible to observe various levels of SNR in the utterances. This imitates the *cocktail party* scenario.

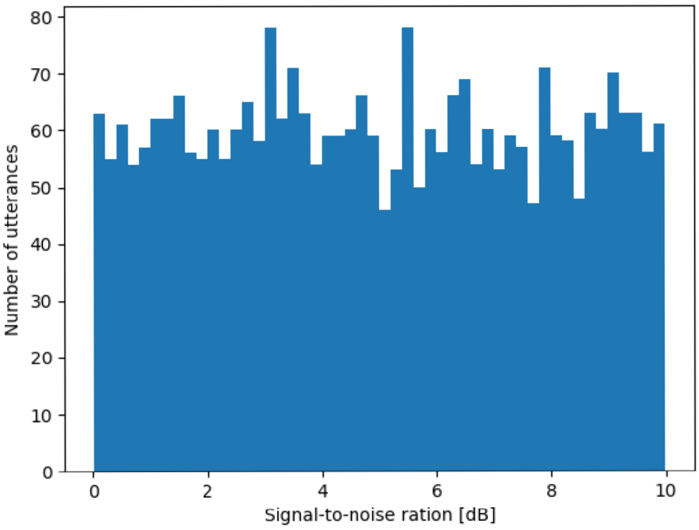

Figure 4.2: Histogram of SNR of one speaker over the other from test dataset.

Chapter 5

Implementation

This chapter describes an implementation of the described neural network elements, tools used for development, and overall approach to the task. As mentioned in the introduction, the model, shown in figure 5.1, is inspired by [3] and uses some alternative approaches. As first, the chapter describes PyTorch [19], a machine learning library used to implement the model. As next, it describes individual parts of the implemented system.

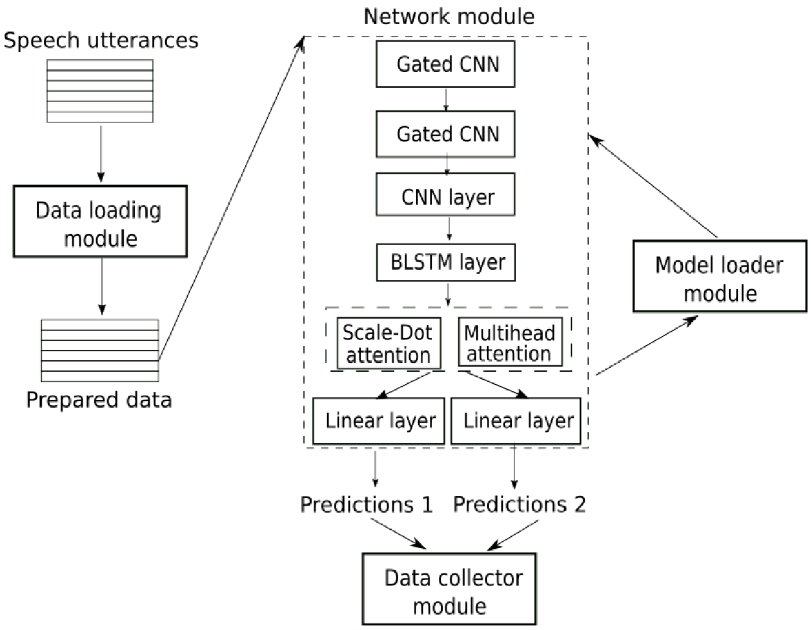

Figure 5.1: Scheme of the implemented architecture modules

5.1 PyTorch

PyTorch[1] [19] is an open-source machine learning library. It is primarily developed by Facebook's AI Research lab. It has a Python interface along with highly optimized C++ and CUDA[2] core.

[1] https://pytorch.org

[2] https://developer.nvidia.com/cuda-zone

As a basic unit for computation tasks PyTorch uses a *tensor*, an n-dimensional array of the base data types. To perform calculations it uses a computational graph, a graph representing a flow of the data between corresponding tensors representing specific operations to obtain the output. PyTorch can perform its calculations on CPUs and GPUs, where GPUs are used for faster computations.

PyTorch allows special handling and loading of the data through classes Dataset and Dataloader. A Dataset class is used for the preparation of data into a format for class Dataloader. Dataloader is then used to serve data to the model when required, allowing to use features like data shuffling, setting a batch size or mechanism called collate function, a function, or a class executed before serving the particular portion of required data.

Many well known neural network classes like CNN, LSTM, or feed-forward networks are already implemented in this library. This allows creating models and building layers of those types. It also provides utilities needed for training like activation and loss functions. To train a model PyTorch offers a module containing optimizers to update weights of the neurons.

It is also worth mentioning that PyTorch allows saving parameters of the trained model in the form of a Python dictionary. It can also convert it into formats such as ONNX[3].

5.2 Parts of the system

This section contains a detailed description of individual implemented components and their role in the models.

5.2.1 Data collector

A data collector module creates necessary files and folders for the data collection of the training, cross-validation, and evaluation process. These files are always created in a symbolic path from which the system is executed. It collects total accuracy and total loss from every epoch as well as the accuracy and loss of every batch in the training and validation processes. The collected data are stored in the form of a Python list with a Python module called **Pickle**[4].

5.2.2 Data loader and preparation

A data loader module uses a PyTorch classes Dataset and Dataloader, mentioned in section 5.1 above. With the use of the PyTorch class Dataset, a module loads data from provided files and then prepares the loaded data into a format for the system to process. The datasets are composed of various acoustic feature vectors with variable lengths, as can be seen in figure 5.2. However, to process them they all must be of the same length.

To achieve this, the module has at its disposal two different approaches available. First consists of an use of zero-padding where all sequences and corresponding targets with shorter lengths than the longest one from a batch are padded with zero frames to fill in the missing space, shown in figure 5.3.

The targets are padded with frames containing a value equal to -100 because it serves for PyTorch as an indicator for overlooking the padded parts of sequences during the calculation

[3] https://onnx.ai/

[4] A module used for saving and loading Python object structure, https://docs.python.org/3/library/pickle.html

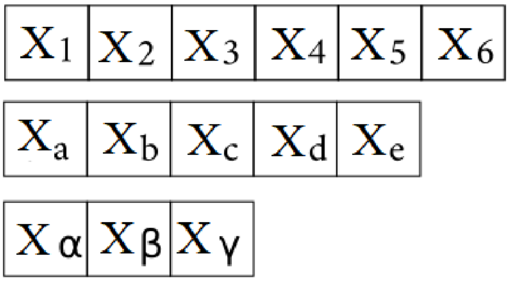
X_1 X_2 X_3 X_4 X_5 X_6
X_a X_b X_c X_d X_e
X_α X_β X_γ

Figure 5.2: Original acoustic feature vectors

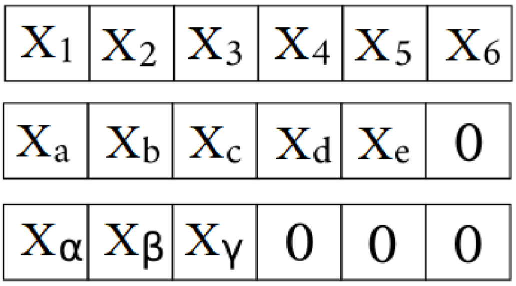
X_1 X_2 X_3 X_4 X_5 X_6
X_a X_b X_c X_d X_e 0
X_α X_β X_γ 0 0 0

Figure 5.3: Zero-padded acoustic feature vectors

of the loss. It is important to realize this, otherwise, the model would include them into the learning process and they could corrupt it.

This approach, while sufficient, carries with itself a complication though. It is very demanding on computational resources because it creates padded sequences based on the length of the longest one and it can occupy a considerably large part of the memory. This situation is solved by the second approach. The approach in some ways is similar to the first one. It also creates frames for padding, however, these frames are not omitted. Furthermore, this method divides the input sequences and targets into smaller fragments of the selected size.

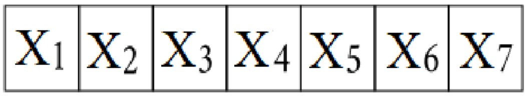

Figure 5.4: Uncut original speech sequence

Vectors smaller than the selected size of the cut are dropped. In the case of vectors bigger than the size of the cut there almost always remains a piece smaller than the size of the cut. This piece is not dropped, however. Instead, the module cuts an additional part from the particular sequence to fill missing frames. The module obtains this additional part by selecting the appropriate point in the particular input sequence, from which the number of frames is equal to the selected size of the cut. This part is then separated and added to the others. This process is illustrated in figure 5.5 where the original sequence from figure 5.4 was divided into pieces with a size of the cut being equal to three. This approach minimizes the memory requirements as well as the required computational power.

To explain why the smaller sequences than the size of the cut are dropped is that they could create invalid conjunction because there is no valid part to fill the missing space.

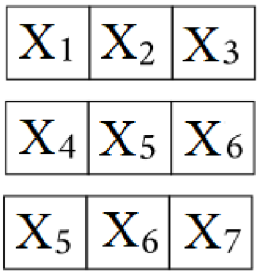

Figure 5.5: Sequence-padded acoustic feature vectors with size of the cut being equal to three.

The size of the cut, however, must be selected wisely as if it is too small, the model might not learn continuity patterns from the cut sequences and if it is too large, it might suffer from the same problems as the zero-padding method.

5.2.3 Network modules

As mentioned at the beginning of the section, the implementation contains two modules with different sizes. The first one has a size that is smaller when compared to the model variations in [3], at least in the case of the BLSTM layers. and then the larger one. It allows choosing between the models depending on the available computational resources.

The module is capable to run three different modes: 1. with Scaled Dot-Product attention, 2. with Multihead attention and 3. run without attention. When running in the first two modes, the module can save attention weights from selected attention mechanism in the form of **NumPy**[5] arrays for further analysis. The attention mechanisms are implemented as Python classes. If the mode without attention mechanism is selected, the mechanism part is substituted by the linear layer. The module produces two outputs for each speaker.

5.2.4 Model loader module

This module is responsible for saving and loading the current state of the network model. It saves the state depending on the selected type of model. There are in total of two types of saving model states: 1. saving of the best state - the state of the model is saved based on the collected statistics. The calculated train loss from the particular iteration must be lower than it was during the iteration when the model produced the best results and the same condition must be satisfied in case of the cross-validation loss. This prevents saving overfitted states and helps to preserve the best performance. 2. saving of the next state - the state of the model is saved directly after the completion of the particular training iteration. It saves them separately, thus allowing more experimenting with different states. Similarly, it allows the loading of the desired status of the model, the best, or from the last iteration.

[5] A library written in Python designed for scientific computing, https://numpy.org

5.2.5 Training, cross-validation, and test modules

These modules start the desired process of training or validation of the model on selected datasets. The training module comes in two different versions. Each one will launch the training of the selected model of the corresponding size. The cross-validation process is launched by the training module every time after each training epoch. Then depending on the selected saving mode it decides whether to save the current status of the model. The test module is capable to save more to save can also save predictions of the model for both speakers in the form of Numpy arrays or accuracies of produced predictions for individual utterances identified by their name in the form of the Python dictionary.

All modules use a PIT cost function to determine the best permutation of the speakers and then calculate corresponding accuracies, although they do not scale final loss as in [3] due to experiments with various learning rates. If input sequences were zero-padded these padded frames are ignored during the calculation of the accuracy. If it is possible, the modules switch automatically to perform the computations on one GPU possessing CUDA.

Chapter 6

Experiments

This chapter describes performed experiments to evaluate the efficacy of the implemented model. There were inspected various architectures of the attention modules. First, there are introduced metrics applied to evaluate the overall performance of the model. Then chapter describes the evaluation dataset used for experiments and after that, it describes experiments with each architecture.

6.1 Metrics for model evaluation

To evaluate the performances of the individual models the following metrics were used:

- **Prediction accuracy** - This metric takes a total number of correct predictions of frames with respect to the permutation with the lowest loss by the total number of frames in each iteration. If there is zero-padding used, the model will automatically exclude the padded frames.
- **Word error rate** - Word error rate (WER) [1] is an evaluation metric, used to evaluate the performances of ASR systems. Value of this metric defines the total percentage of errors calculated from the alignment of reference words with predicted words produced by ASR system [1]. It is can be defined by following formula [1]:

  $$WER = \frac{I + S + D}{N} \times 100 \tag{6.1}$$

  where I represents a number of the inserted words, S represents a number of the substituted words, D represents a number of the deleted of the words and N is a number of the reference words.

Apart from listed metrics we also observed the overall losses of training and validation sets to determine possible overfitting or other strange behavior of models. To determine, if the particular attention mechanism works properly there were analyzed produced attention weights of that mechanism.

6.2 Neural network models

This thesis explores the performances of three different implemented models. Previous chapters described various components. These three models combine these to determine their efficacy in dealing with the problem of overlapping utterances.

As mentioned in the introduction, the implemented models differ from the proposed ones. They share almost the same elements from previous chapters with the model variations from the original article [3]. But they are all of the different sizes, an order of those elements is different from the variations from [3]. And the major difference comes with the attention mechanism. The implemented models use the attention mechanism inspired by the article [24], explained in 2.5, different from the proposed one. Although Multihead attention differs slightly from the original [24], explained below. The different size is used mainly due to the extension of the dataset and shortage of computational resources. However, as mentioned in chapter 5, the system does contain also large-sized variations for experimenting if there is a sufficient number of resources. However, these variations were not were used in the experiments.

Apart from the attention mechanisms modules, the variations use a *Leaky ReLU* as the activation function. It is a special modification of the ReLU function created to compensate its drawback in the form of not learning when the activation is equal to zero [7]. To adjust the network weights with gradient descent the models use **Adam optimizer** [15].

All models share the same number of two gated convolution layers. The first layer is comprised of two CNN layers, each of size 20 and in the second the CNN layers have a size of 100. After them succeeds the convolutional layer. This layer is comprised of 200 neurons. In the end, there are two output linear layers of size 3368 which is equal to the total number of classes. The rest of the architecture differs in each of the models. These differences are explained in the following subsections.

The pre-trained systems first produce predictions and save them in the form of a Numpy array along with a collection of the loss and accuracy statistics from test validation. After the completion, predictions are then evaluated which will then evaluate the overall level of WER of the particular model for both speakers.

6.2.1 The model without attention mechanism

This model variation does not possess any of the attention mechanisms described in section 2.5. Instead, it, as can be seen in figure 6.1 contains two BLSTM layers, each one having a hidden equal to 200 and a linear layer of size 516. This layer does not separate the predictions as it is in case of attention mechanisms. The separation is done by output layers. The number of BLSTM layers is higher than in the case of the other variations.

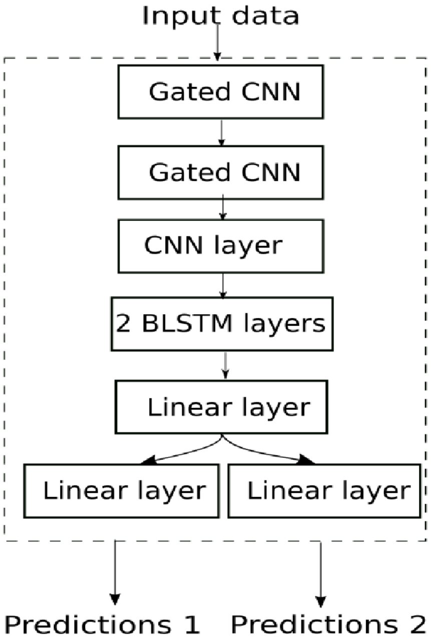

Figure 6.1: The model architecture without attention mechanism with individual layers.

6.2.2 The model using Scaled Dot-Product attention mechanism

In this model variation, shown in figure 6.2, the linear layer from the previous variation is substituted by the *Scaled Dot-Product* attention layer. This layer first uses three linear layers of size 200 to produce three different outputs for the creation of keys, queries, and values. Before this model performs non-linear transformation with $ReLU$ activation function over these outputs. Then they are sent into the next three linear layers of size 516 to produce keys, queries, and values sets. This process is performed over the input data for each speaker separately. It also contains only a single BLSTM layer where its hidden size is equal to 200, mainly due to a lack of computational resources.

6.2.3 The model architecture using Multihead attention mechanism

This model variation, shown in figure 6.3, as the previous from subsection 6.2.2 also contains a single BLSTM layer where its hidden size is equal to 200. uses the same process to produce keys, queries, and values. It, however, uses the Multihead approach explained in 2.5. There are three *heads* for one speaker, thus six for both, each consisting of the set of six linear layers of size 172. The first three produce the output more fitted for the creation of attention vectors. They use also use $ReLU$ activation function for transformation. This is the difference between the Multihead attention mechanism from [24] and the implemented model. In [24], there was no activation function used in attention module. The other three produce the keys, etc. Then their outputs are concatenated and sent to another parametric linear layer of size 516. This process is also performed over the input data for each speaker separately.

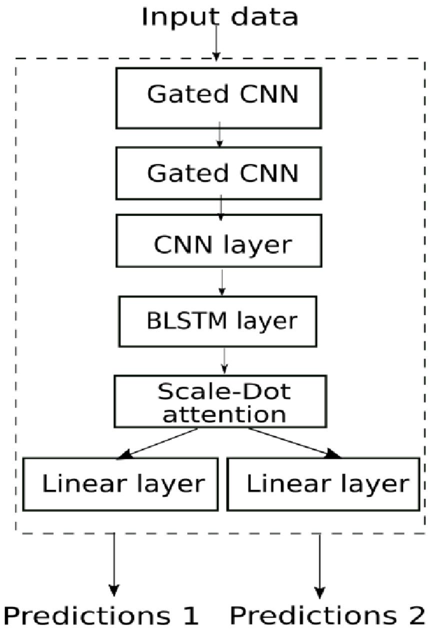

Figure 6.2: The Scaled Dot-Product architecture model scheme with individual layers.

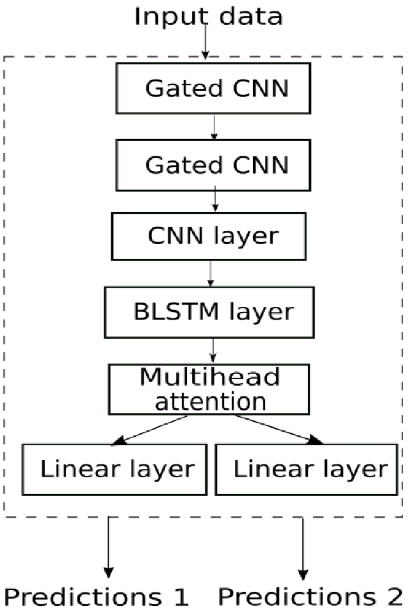

Figure 6.3: The Multihead architecture model scheme with individual layers.

6.3 Model experiments

To train and evaluate model variations were from the described datasets extracted 40-dimensional acoustic features and corresponding targets using **Kaldi** [1] toolkit [20]. There are in total 3386 possible output classes.

All of the described models from above were trained on one GPU and each one was trained for 200 epochs. The following subsections contain detailed descriptions of experiment results conducted on the test evaluation dataset. There were tested various learning rates for each model. The final results of the best experiments can be found in table 6.5. The following subsections describe the experiments with each model group. All the figures containing statistics from experiments in these subsections were created from performances of the best model instances of each variation.

WER on test dataset was also evaluated using another provided evaluation program using the Kaldi toolkit. Other statistics were collected directly by the model itself.

6.3.1 Experiments with the model without attention mechanism

Despite having the simplest structure from variations this model showed results beyond expectations. As can be seen in figure 6.4 in the first picture, the training loss tends to decrease and training accuracy also increases as can be observed in the second picture of figure 6.4, so the model trained correctly. However, in the case of cross-validation loss in the first picture of figure 6.4, it is possible to observe a tendency of a curve to grow in an increasing number of epochs. In the case of cross-validation accuracy in the second picture of figure 6.4 it grows but then it settles in level around 45%. These are signs of overfitting.

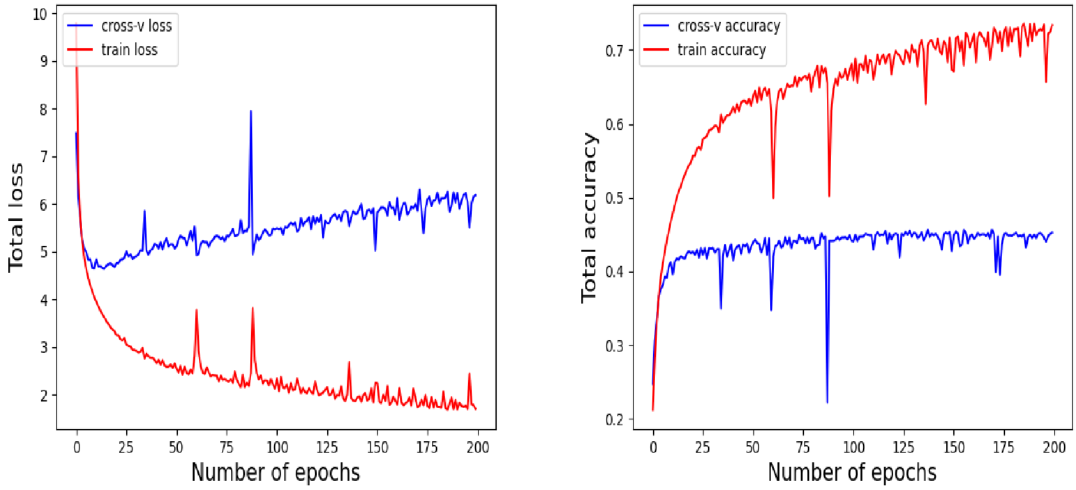

Figure 6.4: On the left, progress of total train and cross-validation loss from each epoch, On the right, progress of total train and cross-validation accuracy from each epoch of the best instance of no attention model.

[1] A toolkit used for speech recognition, written in C++, https://kaldi-asr.org/

In table 6.1 is possible to observe all experiments performed with no attention model.

Learning rate	WER value	Avg. frame accuracy
0.001	28.4%	46.3%
0.0006	32.5%	46.5%

Table 6.1: WER level, learning rate and average frame accuracy of each model of no attention variation.

The model with learning rate being equal to 0.001 produced very good results which even exceeded all model variations without attention mechanism and even models with attention mechanism from article [3].

Because experiments with this model produced the very best results from all performed experiments, it is worth to evaluate regarding this thesis the possible influence of the length of utterances and SNR to the accuracy of predictions.

The effect of length utterances of the accuracy of predictions of frames is captured in figure 6.5. It can be observed a certain level linear dependency between them which was proved by calculation of the **Pearson correlation coefficient**, a value indicating how much two variables are dependent on each other. Its value equals **0.24666169** which indicates a positive correlation, which indicates a certain level of dependency between the accuracy of the prediction and the lengths of the utterances.

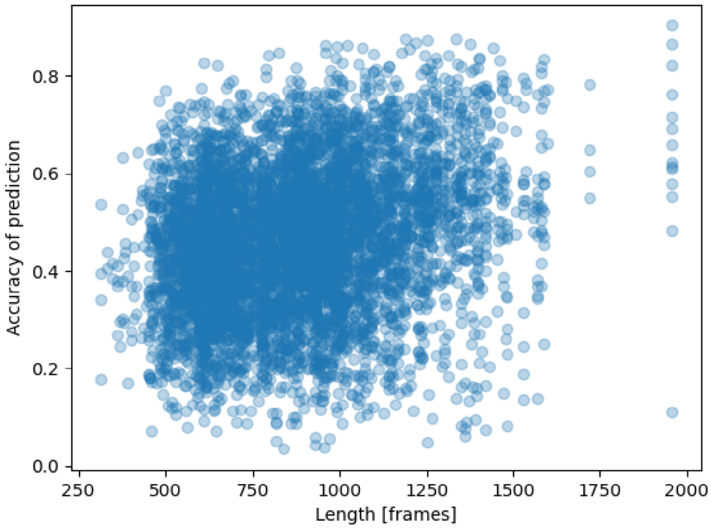

Figure 6.5: Dependency of the accuracy of predictions on length of the utterance of test dataset captured from the best model performance.

In the case of the influence of the SNR on the accuracy of predictions, the correlation coefficient is equal to **0.1392165**. This also indicates a positive correlation and a certain level of dependency between the accuracy of the prediction and the SNR of the utterances. The effect is displayed in the following figure.

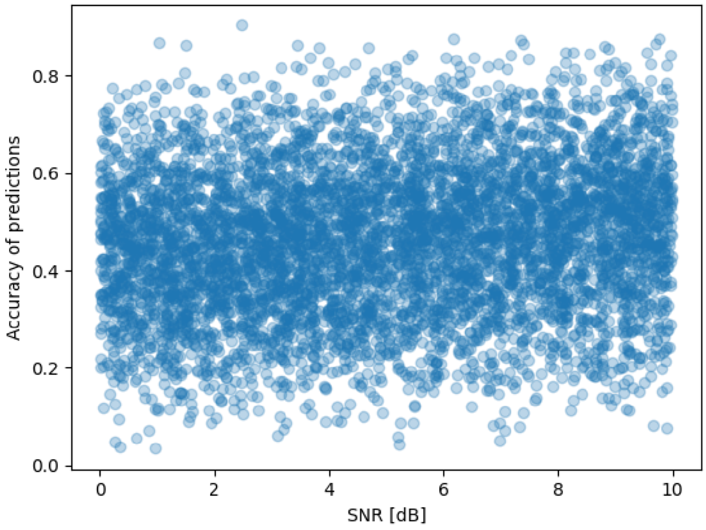

Figure 6.6: Dependency of the accuracy of predictions on the SNR of the utterances.

In following table are average accuracies of frame predictions for each group of speakers based on their gender:

Gender	Avg. frame accuracy
Women	48.1%
Men	43.9%
Mixed	46.9%

Table 6.2: Average accuracies of frame predictions for each group of speakers based on their gender.

6.3.2 Experiments with Scaled Dot-Product attention mechanism model

Experiments conducted on this model variation also produced interesting results. The use of the attention mechanism showed that it learns correctly. The overfitting problem is also apparent, however, with a closer look at figure 6.7 it is possible to observe that loss of Scaled Dot-Product variation does not grow so rapidly as in case of no attention variation meaning the attention model tends to get overfitted slowly. Still, it didn't perform as well as its predecessor in terms of accuracy and WER. Here, in figure 6.7 on the right picture, the accuracy level stops around 40% and even training accuracy is worse than in case of no attention model but is very close to it.

In figure 6.8 is displayed an array containing the attention weights from the utterance which scored the highest accuracy. It is possible to observe the attention weights of the mechanism over concentrated along the main diagonal across the time axes. Figure 6.9 displaying the utterance with worst scored accuracy also shows a major concentration of the attention weights along the main diagonal meaning the attention mechanism works correctly even in this case.

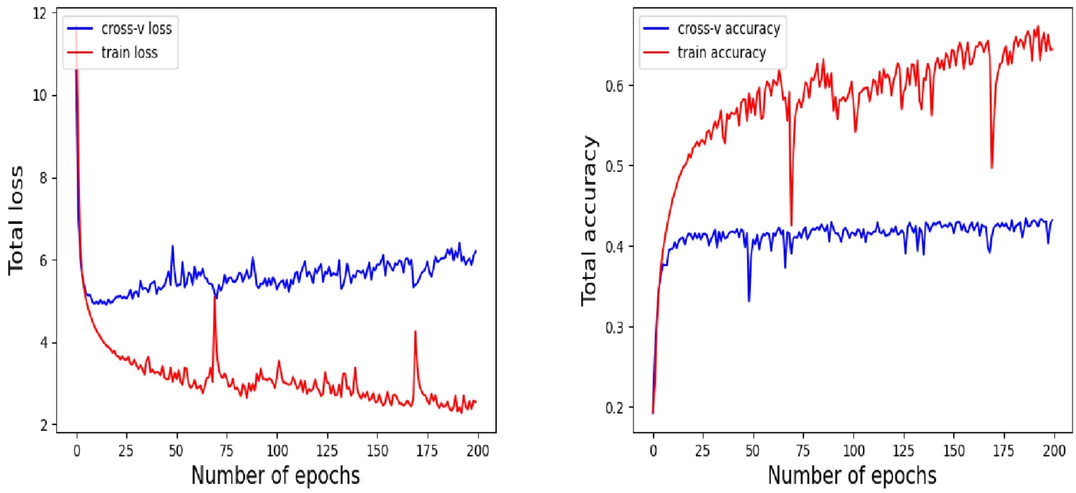

Figure 6.7: On the left, progress of total train loss from each epoch, On the right, progress of total cross-validation loss from each epoch of the best instance of Scaled Dot-Product model.

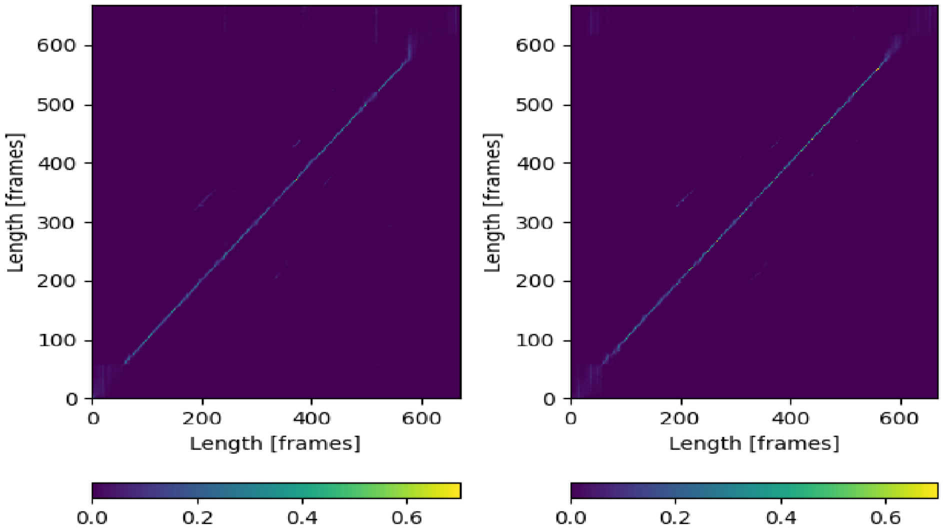

Figure 6.8: Attention weights of utterances of the best instance of Scaled Dot-Product model.

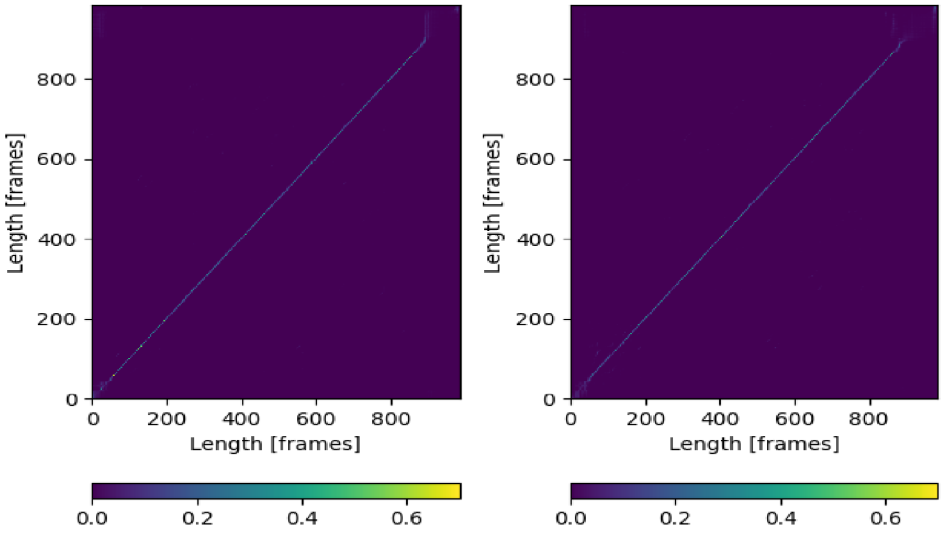

Figure 6.9: Attention weights of utterances of the best instance of Scaled Dot-Product model.

The following table shows the results of the performed experiments.

Learning rate	WER value	Avg. frame accuracy
0.001	29.8%	46.0%
0.0008	31.1%	44.9%
0.0004	36.4%	37.4%

Table 6.3: WER level, learning rate and average frame accuracy of each model of Scaled Dot-Product attention variation.

From this table is possible to see that model with learning rate being equal to 0.001 produced the best results. It also exceeded all model variations without attention mechanism and variations with attention mechanism from article [3].

6.3.3 Experiments with Multihead attention mechanism model

The overall results of the conducted experiments showed that the Multihead attention mechanism does not outperform the performance of other model variations. Instead, it produces quite strange levels of WER, shown in table 6.5. Only the model with learning being equal to 0.001 produced reasonable WER. In terms of accuracy, as can be seen in figure 6.10, the model performs similarly to the Scaled Dot-Product model in cross-validation accuracy. Observing figure 6.10, cross-validation loss once again shows the overfitting of the model as did previous models. The curve of the cross-validation loss is more stable in comparison with no attention cross-validation loss and is more similar to the one of the Scaled Dot-Product model, i.e. it does not grows so quickly. Although it produces the highest losses from all variations.

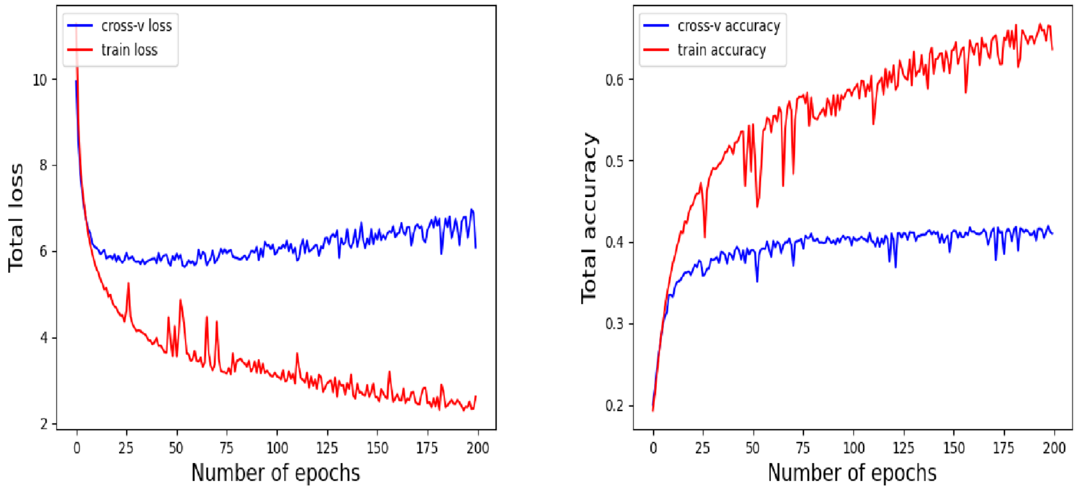

Figure 6.10: On the left, progress of total train loss from each epoch, On the right, progress of total cross-validation loss from each epoch of the best instance of Scaled Dot-Product model.

The most interesting aspect to watch is, however, the distribution of attention weights of the individual heads. From figure 6.11 can be seen the attention of the individual heads of best prediction, each shows to have a different distribution of the weights over the matrix. Weights in the second and third head show a certain form of attention along the main diagonal, however, not as clear as in the case of the Scaled Dot-Product model. They all have a different distribution, proving that each one is concentrating on other parts of the input sequence. In figure 6.12 are heads containing weights from utterance with worst accuracy, they also show different distributions and in the first two heads, there are more concentrated brighter regions than in the case of utterance best accuracy. This might indicate that in case of this utterance heads attend more to the wrong parts of input information.

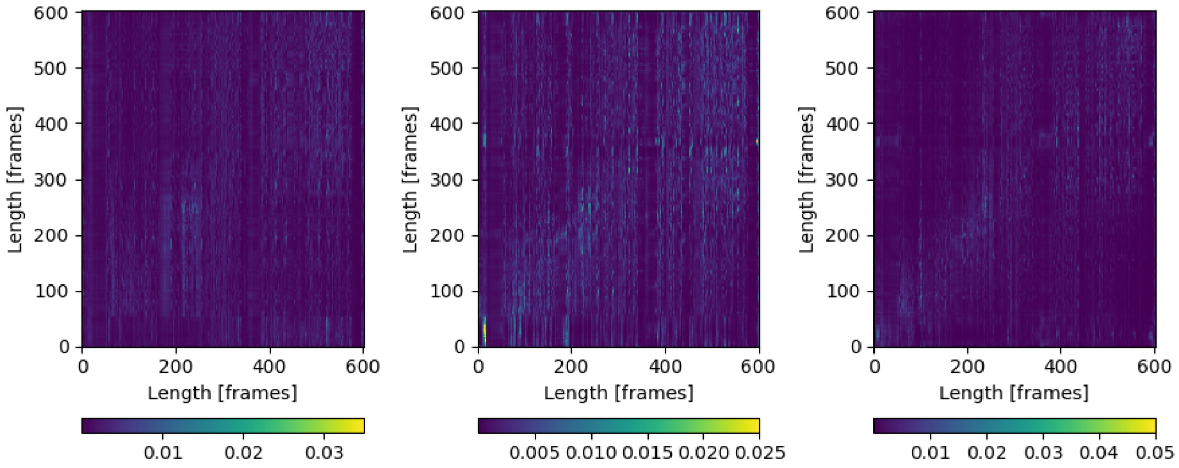

Figure 6.11: Attention weights of individual heads from the best-scoring utterance of the best instance of Multihead model.

In the following table is possible to observe the results of the individual experiments. It only outperformed two variations with no attention with the worst WER mechanism and variation with attention mechanism without Gated convolutions with worst WER equal to 34.6% from [3].

Learning rate	WER value	Avg. frame accuracy
0.001	34.5%	40.4%
0.0006	40.8%	42.3%
0.0004	39.6%	38.9%
0.0002	40.6%	38.1%

Table 6.4: WER level, learning rate and average frame accuracy of each model of Multihead attention variation

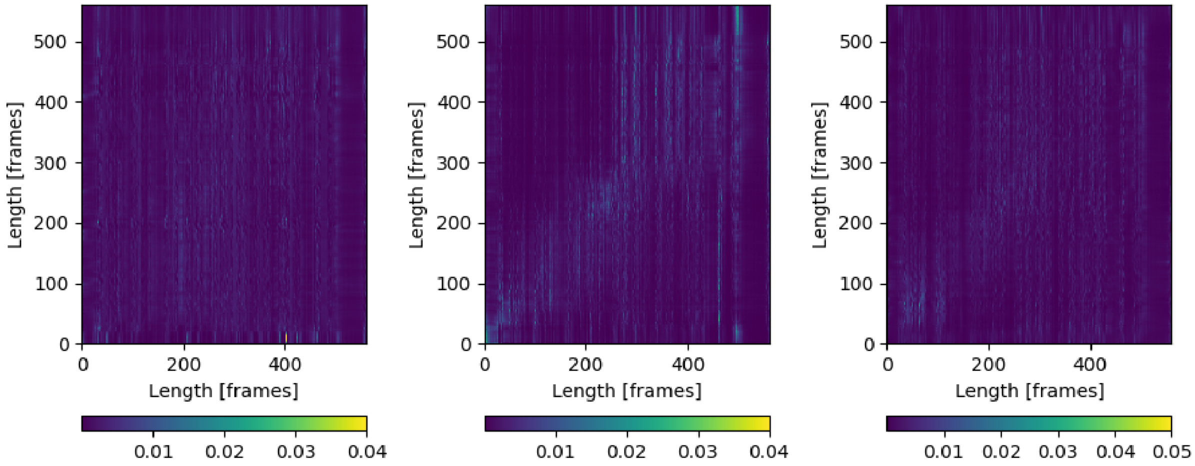

Figure 6.12: Attention weights of individual heads from the worst-scoring utterance of the best instance of Multihead model.

6.4 Summary of experiments

Model type	Learning rate	WER value
No attention	0.001	28.4%
Scaled Dot-Product att	0.001	29.8%
Multihead att.	0.001	34.5%

Table 6.5: Best result of each model variation and its learning rate.

With performed experiments with each model variation, it was demonstrated, as can be seen in the figures 6.4, 6.7 and 6.10 that the case of overfitting was present in all experiments. However, before it started, all models saved their best status to evaluate the WER value. One of the possible explanations of why models are overfitting is because the models weren't optimally tuned as all models use different learning rates. It is possible to observe that when learning is equal to 0.001 the WER level of each model decreases. Moving the level of learning rate only just by one-thousandth of its value can have dramatic results as can be observed in the case of the Scaled Dot-Product attention model. Experiments also showed that higher frame accuracy does not guarantee a better WER value.

It was demonstrated that models using the attention mechanism did not produce better results than the model without it. One of the possible explanations consists of the loss of the context because the utterances were fragmented during the training procedure due to the hardware limitations and because of this, the attention mechanisms maybe did not learn the whole context of where to attend.

What is, however, interesting is the fact that the model using Multihead attention has the worst results as can be seen in table 6.5. This again can be caused by a low number of heads as there are only three heads in the Multihead attention model or that one part of the mechanism uses activation function before producing keys, queries, and values. However, as can be seen in figure 6.11 the Multihead attention does learn some information in each head. Better results may be achieved by increasing the number of total heads or delete activation functions entirely from the Multihead attention mechanism, as they may have affected its performance because they aren't part of the original. Or it may be caused by insufficient size.

The following table shows a comparison between the best results of implemented models and best models from the article [3].

Model type	WER value
Implemented	
No attention	28.4%
Scaled Dot-Product att.	29.8%
Multihead att.	34.5%
From the article	
No attention	32.7%
With attention mech.	31.0%
With attention mech. + gated conv.	31.6%

Table 6.6: Comparison of best results of implemented models with best models from article [3].

As can be seen in table 6.6, the implemented no attention model outperformed all best-scoring model variations from [3].

Also, Scaled Dot-Product attention variation produced reasonable WER results. It also outperformed the best model variations from [3] and it also outperformed the Multihead attention which might indicate that it is better suited for this type of task, but this might change with the possible modifications of the model mentioned above. Also, variations with attention mechanisms had only one BLSTM layer which might have some possible influence on the final performance.

There might be a suggestion that the system proved to be better due to lower a lower number of output classes than in the original article. However, the difference is not that huge as 3368 output classes aren't much greater than 3429 which is a number of output classes from [3]. It might have affected the total accuracy of the predictions of frames, however, it does not affect the WER as the same words were used to get the WER level.

6.4.1 Possible improvements

This subsection contains a list of possible improvements of the implemented models for future contexts:

- **Tuning of parameters** - as shown in tables of performances of each variation, the learning rate can have a great impact on the final results of this model. Because the experiments are quite expensive to perform, there aren't results for a wider range of learning rates. There is also the possibility to experiment with the size of the individual layers and their order in the model.

- **Application of overfitting countermeasures** - all types of models suffer from overfitting. Regularization techniques are key to prevent this, especially in the case of the larger models. It might be also the key for the attention mechanism to work more properly as a large-sized model along with regularization techniques could perform better because in experiments it was shown that the model learned with it some patterns and the layers would prevent the overfitting. Data augmentation is not the way as there is sufficient data already present in used datasets and adding more would only cost more time to train the model and more resources. The early stopping is already partially applied, i.e. models are capable to save their best states.

Chapter 7

Conclusion

The main objective of this thesis was to implement neural network architecture using the combination of the described components and technique used to improve the multi-talker speech recognition described by the [3]. There were built three different model variations, one without attention mechanism and two with it. The variations using the attention mechanism have each one a different type of this mechanism. Both mechanisms are based on [24]. They are different from the attention mechanism used in [3].

Implemented models were then trained and evaluated on the standard datasets introduced by [9]. On the implemented model variations were then conducted experiments to explore their capabilities. The models proved to produce good results, especially Scaled Dot-Product attention variation and no attention variation.

The performed experiments showed, that the model without attention mechanism can produce better results than those with it. The best no attention model with WER being 28.4% outperformed the best result from [3] by 2.6%, where the best model which contains attention mechanism produced WER 31.0%. This shows that in this type of task models with the attention mechanism did not outperform the model without it.

However, the Scaled Dot-Product attention model is very close to the best result of the no attention model with WER being 29.8%. The difference between their best WER results is only 1.4% and Scaled Dot-Product attention also outperformed the best result from [3] by 1.2%.

Multihead attention model did not perform so well as did the first two. Its best WER result was 34.5% which is not better than the best result from [3]. This, however, might also be the result, that Multihead attention is not the entirely same as in [24] or that there is a low number of heads, etc.

However, there is also some space to improve the performance as suggested in 6.4.1, especially in the case of Multihead attention. The mechanism might also improve its performance with the larger model. However, because it was proved that all model variations suffer from overfitting phenomenon, the first step to improve them is to apply the regularization techniques. Then, if they improve, it will depend on further experimenting with the individual parameters of each model. And because models showed certain potential it might be interesting to explore other possibilities of these architectures.

Bibliography

[1] Ali, A. and Renals, S. Word Error Rate Estimation for Speech Recognition: e-WER. In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, July 2018, p. 20–24. DOI: 10.18653/v1/P18-2004. Available at: https://www.aclweb.org/anthology/P18-2004.

[2] Bishop, C. M. et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[3] Chang, X., Qian, Y. and Yu, D. Monaural Multi-Talker Speech Recognition with Attention Mechanism and Gated Convolutional Networks. In: *Proc. Interspeech 2018*. 2018, p. 1586–1590. DOI: 10.21437/Interspeech.2018-1547. Available at: http://dx.doi.org/10.21437/Interspeech.2018-1547.

[4] Gales, M. and Young, S. The application of hidden Markov models in speech recognition. *Foundations and trends in signal processing*. Now Publishers Inc. 2008, vol. 1, no. 3, p. 195–304.

[5] Garofolo, J. S. et al. *CSR-I (WSJ0) Complete LDC93S6A. Web Download. Philadelphia: Linguistic Data Consortium*. 1993.

[6] GERS, F. *Long Short-Term Memory in Recurrent Neural Networks*. 2001. Dissertation. Universität Hannover.

[7] Goodfellow, I., Bengio, Y. and Courville, A. *Deep learning*. MIT press, 2016.

[8] Graves, A. and Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks*. Elsevier. 2005, vol. 18, 5-6, p. 602–610.

[9] Hershey, J. R., Chen, Z., Le Roux, J. and Watanabe, S. Deep Clustering: Discriminative Embeddings for Segmentation and Separation. In: *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. March 2016, p. 31–35. DOI: 10.1109/ICASSP.2016.7471631. Available at: https://www.merl.com/publications/TR2016-003.

[10] Hinton, G., Deng, L., Yu, D., Dahl, G. E., Mohamed, A.-r. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal processing magazine*. IEEE. 2012, vol. 29, no. 6, p. 82–97.

[11] Huang, X. and Deng, L. *An Overview of Modern Speech Recognition.* Handbook of Natural Language Processing, Second Edition, Chapter 15 (ISBN: 1420085921)th ed. Chapman & Hall/CRC, January 2010. 339-366 p. Available at: https://www.microsoft.com/en-us/research/publication/an-overview-of-modern-speech-recognition/.

[12] Isik, Y., Le Roux, J., Chen, Z., Watanabe, S. and Hershey, J. R. Single-Channel Multi-Speaker Separation using Deep Clustering. In: *Interspeech.* September 2016, p. 545–549. DOI: 10.21437/Interspeech.2016-1176. Available at: https://www.merl.com/publications/TR2016-073.

[13] Johnson, D. H. Signal-to-noise ratio. *Scholarpedia.* 2006, vol. 1, no. 12, p. 2088. DOI: 10.4249/scholarpedia.2088. revision #126771.

[14] Keller, F. *Computational Foundations of Cognitive Science: Lecture 15: Convolutions and Kernels.* 2010. [Online, Cited: 2020-02-26]. Available at: http://www.inf.ed.ac.uk/teaching/courses/cfcs1/lectures/cfcs_l15.pdf.

[15] Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *ArXiv preprint arXiv:1412.6980.* 2014.

[16] LeCun, Y., Bengio, Y. et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks.* 1995, vol. 3361, no. 10, p. 1995.

[17] Nielsen, M. A. *Neural Networks and Deep Learning.* Determination Press, 2015.

[18] Olah, C. *Understanding LSTM Networks.* Aug 2015. [Online, Cited: 2019-01-19]. Available at: https://colah.github.io/posts/2015-08-Understanding-LSTMs/#fnref1.

[19] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H., Larochelle, H., Beygelzimer, A., Alché Buc, F. d; Fox, E. et al., ed. *Advances in Neural Information Processing Systems 32.* Curran Associates, Inc., 2019, p. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[20] Povey, D., Ghoshal, A., Boulianne, G., Burget, L., Glembek, O. et al. The Kaldi speech recognition toolkit. In: IEEE Signal Processing Society. *IEEE 2011 workshop on automatic speech recognition and understanding.* 2011, CONF.

[21] Reynolds, D. A. Gaussian Mixture Models. *Encyclopedia of biometrics.* Berlin, Springer. 2009, vol. 741.

[22] Samudravijaya, K. *Automatic Speech Recognition.* 2004. [Online, Cited: 2020-01-26]. Available at: http://www.iitg.ac.in/samudravijaya/tutorials/asrTutorial.pdf.

[23] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research.* 2014, vol. 15, no. 56, p. 1929–1958. Available at: http://jmlr.org/papers/v15/srivastava14a.html.

[24] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L. et al. Attention is All You Need. In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. Red Hook, NY, USA: Curran Associates Inc., 2017, p. 6000–6010. NIPS'17. ISBN 9781510860964.

[25] Weng, L. Attention? Attention! *Lilianweng.github.io/lil-log*. 2018. [Online, Cited: 2020-02-27]. Available at: http://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html.

[26] Yu, D., Kolbæk, M., Tan, Z. and Jensen, J. Permutation invariant training of deep models for speaker-independent multi-talker speech separation. In: *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2017, p. 241–245.

Appendix A

Content of attached CD

- **src** folder - folder containing all source files of neural network models.
- **pre-trained_models** folder - folder containing pre-trained neural network models.
- **Hradil_bachelor_thesis_latex.zip** file - file containing source code for Latex to generate bp.pdf.
- **bp.pdf** file - file containing pdf file of the bachelor thesis.
- **README.txt** file - file containing detailed description of individual items on CD and guide how to use network scripts.