Journal of Theoretical and Applied Information Technology
29th February 2020. Vol.98. No 04
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

AUTOMATIC SPEECH RECOGNITION SYSTEM FOR KAZAKH LANGUAGE USING CONNECTIONIST TEMPORAL CLASSIFIER

[1,2]YEDILKHAN AMIRGALIYEV, [1,3]DARKHAN KUANYSHBAY, [1,4]DIDAR YEDILKHAN, [1,3]SHOIYNBEK A

[1]Institute of Information and Computational Technologies CS MES RK, Kazakhstan

[2]Al-Farabi Kazakh National University, Kazakhstan

[3]Suleyman Demirel University, Kazakhstan

[4]Astana IT University, Kazakhstan

E-mail: [1]amir_ed@mail.ru, ainur79@mail.ru

ABSTRACT

This scientific report illustrates the performance evaluation of the well-known, recently popular neural network Connectionist Temporal Classifier (CTC) for speech recognition. The CTC contains LSTM layers with 256 cells and Momentum Optimizer with learning rate 0.005 and momentum 0.9. Dataset that we have used has 35 native speakers with 360 utterances. For expanding the size of our dataset with overall performance augmentation techniques has been applied using Adobe Audition software, which output 20 more speakers to our original dataset. The result of our experiment has been evaluated with LER (Label error rate). LER measures the inaccuracy between predicted an actual texts. The output of the experiment reported training LER 0.000 and validation LER 0.5.

Keywords: *Recurrent Neural Network, Language Model, Acoustic Model, CTC, Data Augmentation, Time Warping.*

1. INTRODUCTION

Today's speech-recognition system always has been developed based on statistical requirements. A generative statistical model (source channel model) results in problems in speech recognition area. As shown in Figure 1, the speaker first thinks of a word sequence W in the mind, after which this sequence passes through speaker's text generator. This sequence enters to communication channel component through his speech generator and signal processing component before passing through a decoder. After, the decoder decodes the created acoustic wave signal to the word sequence.

Decoder

The RNN encoder-decoder is a neural network model that directly computes the conditional probability of the output sequence given the input sequence without assuming a fixed alignment, i.e. P(y1, . . . , yO|x1, . . . , xT) where the lengths of the output and the input, O and T respectively, may be different. For speech recognition, the input is usually a sequence of acoustic feature vectors, while the output is usually a sequence of class indices corresponding to units such as phonemes, letters, HMM states, or words. The idea of the encoder-decoder approach is that for each output yo, the encoder maps the input sequence into a fixed-length hidden representation co, which is referred as context vector. From the previous output symbols and the context vector, the decoder computes

$$P(y_{1,\dots,}y_O|x_{1,\dots,}x_T) = \prod_{o=1}^{O} P(y_o|y_1, \dots, y_{o-1}, c_o).$$

Since the probability $P(y_{1,\dots,}y_O|x_{1,\dots,}x_T)$ is conditioned on the previous outputs as well as the context vector, an RNN can be used to compute this probability which implicitly remembers the history using a recurrent layer.

Let yo be a vector representation of the output symbol yo, where yo is a one-hot vector indicating one of the words in the vocabulary followed by a neural projection layer for dimension reduction. The posterior probability of yo is computed as

$$P(y_o|y_1, \dots, y_{o-1}, c_o) = g(y_{o-1}, s_o, c_o)$$
$$s_o = f(y_{o-1}, s_{o-1}, c_o),$$

where so denotes the output of a recurrent hidden layer f(·) with inputs yo−1, so−1, and co. g(·) is a softmax function with inputs yo−1, so and co. We condition both f(·) and g(·) on the context vector to encourage the decoder to be heavily reliant on the context from the encoder. The previous output yo−1 is also fed to the softmax function g(·) to capture the bigram dependency between consecutive words. We have also investigated a simpler output function without the dependence on the previous output yo−1, i.e. P(yo|y1, . . . , yo−1, co) = g(so, co).

Encoder.As discussed above, the computation of the conditional probability relies on the availability of the context vector co for each output yo. The context vector is obtained from the encoder which reads the input sequence and generates a continuous space representation. The context vector co is obtained by the weighted average of all the hidden representations of a bidirectional RNN (BiRNN):

$$c_o = \sum_t \alpha_{ot} h_t$$

where $\alpha_{ot} \in [0, 1]$ and $\sum_t \alpha_{ot} = 1$; $h_t = (\overrightarrow{h_t}, \overrightarrow{h_t})$ and $\overrightarrow{h_t}, \overrightarrow{h_t}$ denote the hidden representations of xt from the forward and backward RNNs respectively. The context vector co is global, for instance, co = hT. This means the context vector does not depend on the index o, meaning that the whole input sequence is encoded into a fixed vector representation. This approach has produced state-of-the-art results in machine translation when the dimension of the vector is relatively large[9]. When the model size is relatively small, however, the use of a dynamic context vector has been found to be superior, especially for long input sequences.

The weight αot is computed by a learned alignment model for each co, which is implemented as a neural network such that

$$\alpha_{ot} = \frac{\exp(e_{ot})}{\sum_{t'} \exp(e_{ot'})}$$

$$e_{ot} = a(s_{o-1}, h_t)$$

where a(·) is a feedforward neural network that computes the relevance of each hidden representation ht with respect to the previous hidden state of RNN decoder so−1. The alignment model is a single-hidden-layer neural network:

$$a(s_{o-1}, h_t) = V^T \tanh(W s_{o-1} + U h_t)$$

where W and U are weight matrices, and v is a vector so that the output of a(·) is a scalar. More hidden layers can be used in the alignment model.

In the case of using a fixed context vector using an RNN to map the whole input sequence into the context vector is necessary because this vector must represent all the relevant information in the input sequence.

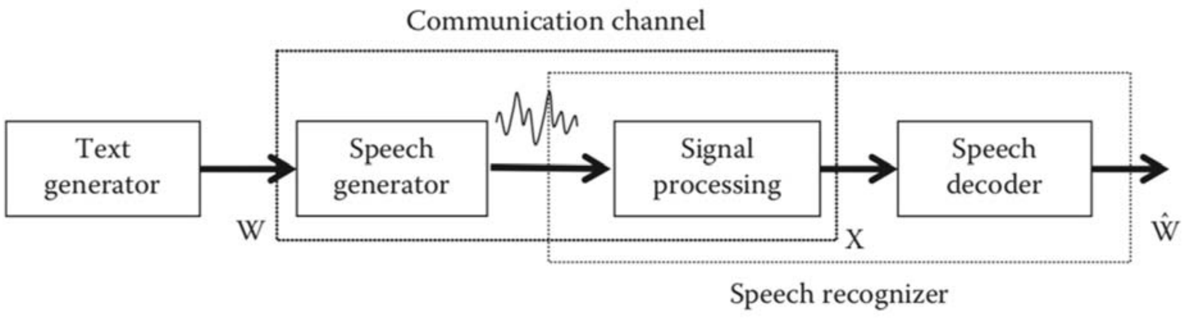


Figure 1. The basic structure of an ASR system

Typical speech recognition system contains the basic elements illustrated in the Figure 2. Applications interact with decoder to get the prediction result that will be adapted to later components. Information about speaker's gender, accents dialects and phonetics are represented by an acoustic model.

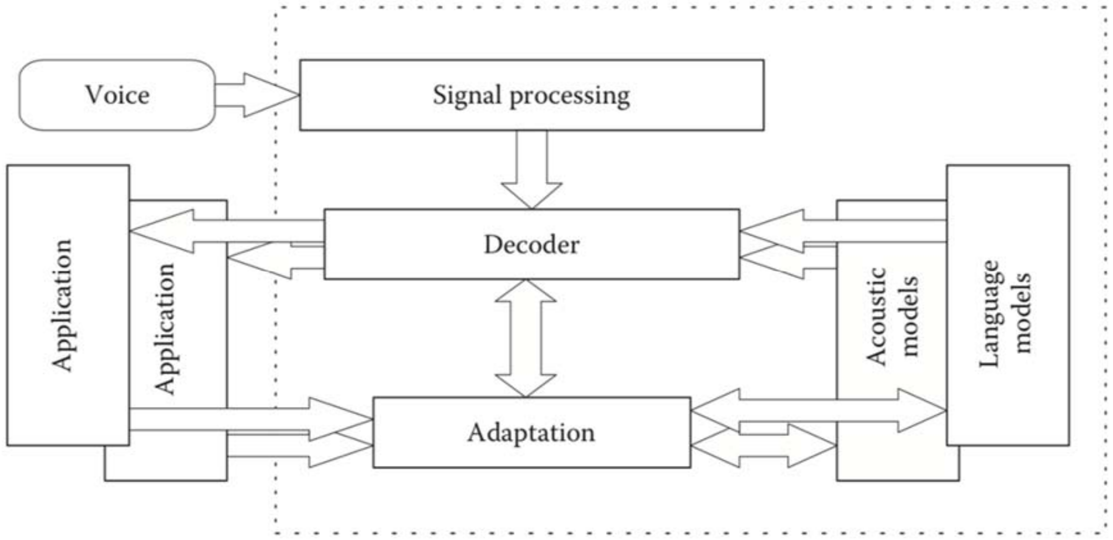


Figure 2. ASR system's architecture

Language models are statistical models that can verify what word is more possible will occur in the given context and what is the probability of the sequence.

Many not fully solved problems in this area related with speaker's attributes, style of speech, the basic speech segments recognition, possible words, unknown words, noise involvement, accents, etc. Well-working ASR system takes over all these problems. The language model represents lexical, grammatical complexity and spoken language variations that defines the acoustic instability of different accents and individual speaker's style of speech.

The recognition process starts with feature extraction of speech signal (audio waveform) taking it to the decoder.

For the given input feature vectors decoder using acoustic and language models generates the words sequence with maximum probability.

All of the described above can be written down by following fundamental equation:

$$\widehat{W} = \arg\max_{w} P(W/A) = \arg\max_{w} \frac{P(W)P(A/W)}{P(A)}$$

where for the feature vector *X*, the aim of ASR system is to generate the word sequence $\widehat{W}$ with a max probability *P(W/X)*. Therefore, the above equation is equivalent to the following:

$$\widehat{W} = \arg\max_{w} P(W)\, P(X/W)$$

where an acoustic model calculates *P(X/W)* and the language model computes P(W). Main challenge is to build a perfect acoustic and language model, which can give the accurate reflection of spoken language. For speech recognition with big vocabulary, we should break word into a sub-word sequence, since the number of words is quite large. *P(X/W)* should consider speaker voice difference, pronouncing difference, difference of environments, and context-dependent phonetic coarticulation differences.

The nature of this process is as follows: in the process of auditory perception by an individual of a speech signal, sub-phonemic units are recognized as separate acoustic elements or phonemes. Taking into account the influence of contextual factors, variations of an individual phoneme in different cases may differ in the specificity of the vibrational parameters due to the coarticulation effect, which determines the effect on the phoneme of the characteristics of other phonemes surrounding it in the speech stream. As a result, the vibrational parameters of the phoneme are dynamic and vary in a given interval, taking into account the peculiarities of the contextual environment of the speech flow. These contextual variations of phonemes are called allophones. In the hierarchical structure of the speech signal, the phonemic level prevails in relation to the allophone level.

This is due to the fact that the phoneme represents a higher level of classification of acoustic signal parameters, characterized by a grouping of contextual variations of the corresponding allophones with subsequent leveling of their noise components and focusing on the unity of vibrational and perceptual qualities.

The highest levels of the hierarchy of the structure of the speech signal are due to transformations of elements in the system “phoneme - syllable - morpheme - word - sentence", which ultimately allow you to create and study semantic relationships between lexical units of speech.

There have been many techniques for recognizing speech. Considering the fact that speech data is complicated in terms of segmentation, it is difficult to build a model with a simple structure. The state-of-the-art technique for ASR (Automatic speech recognition) is always been HMM model [1], which involves other pre-trained models like acoustic model, language model etc. However, recent researches have shown that by using recurrent neural networks [2], we can build such architecture of neural network, which will require only speech data (.wav) and transcription (.txt) to train the model completely, whereas traditional models (HMM) [1] would require data for training language model and acoustic model. This advanced algorithm called Connectionist-Temporal-Classifier [3], the heart of which is RNN. One of the most common and crucial steps in neural network is training. It is important that the model will train fast and at the same time does not over fit or under fit, especially with speech data.

Labeling an unsegmented data is very common and often difficult problem in the sequence-to-sequence models. Straightforward method to solve this is to label each segment of a sequence (for example wave file) manually. However, considering that there are so many words in speech, not counting the sentences, which brings a certain transformations time-consuming, boring and hard to do. To avoid this kind of issues traditional ASR system uses Language model like in [4], which predicts the probability of last word given the sentence and Acoustic model using a progresses like in [5], which gives the phoneme representation of the given speech.

Connectionist temporal classifier [3] requires only a speech data (raw audio) and transcription (txt file) in order to train only one model without involving the Language model. Instead of Language model, it uses dynamic programming method, which called Beam search in [6]. For training the model, any neural network structure uses an optimizer that helps to achieve the good accuracy fast and with no issues (over fitting, under fitting).

The paper is constructed as follows: Section 2 explains how dataset for speech recognition network has been collected and created. Also, it contains how data augmentation techniques and language model have been created from scratch and applied for in our ASR system. Section 3 contains the information about CTC algorithm, Beam search and optimization algorithms, which will be considered in the experiment. Section 4 contains the experiment itself, which is about building a neural network, used optimization algorithms and dataset with a language model. Section 5 illustrates the outcomes of the experiment that shows a result of optimization algorithms comparing with each other (Adagrad, Adam, and Momentum). Section 6 concludes the whole experiment.

2. DATA PREPARATION

Our data for the network has been collected in the base of Suleyman Demirel University. The team consist of 35 people have given the 350 sentences, which have been collected from the famous Kazakh books and news portal. Each person has recorded using Adobe Audition program the utterances and saved it with corresponding transcription file. Since the size of collected dataset is extremely low, we have applied some of the augmentation techniques to extend the size of the current audio data. Instead of increasing the size of dataset using simple augmentation such as changing the pitch, changing the speed, we have applied the modern technique called SpecAugment, which has been presented recently by Google AI team.

2.1. Data Augmentation

There are many important problems and tasks in developing ASR systems based on neural networks. One of such challenges is that these models, having a lot of parameters tend to have over fitting problem. This happens when the model is over-trained on training dataset, but is failing on unseen data. The cause of this problem usually happens when there is not enough data.

In the lack of enough training dataset, it is effective to apply data augmentation to increase the size of original dataset, which has an effective influence on improving the performance of your model. This technique recently often applied on image classification problems. In speech recognition problems, data augmentation method usually involves changing and deformation of an audio wave by adding speed or adding noise to the background. This also helps the network to be much robust and learn relevant features by taking multiple versions of original input. However, existing traditional methods of audio augmentation input takes a bit long time in terms of computation and may require external data.

Instead of applying an augmentation of data as a traditional method, there is a way to apply this technique directly to spectrogram and it's called SpecAugment. This method is simple, computationally cheap and doesn't need any extra additional data. It showed the latest performance on SwitchBoard and LibriSpeech data in Automatic speech recognition tasks.

2.2. Specaugment

In ASR, the given audio is normally represented as spectrogram (Figure-3), before going into the network as an input. Training data augmentation is typically applied before an audio is converted to spectrogram, so that new spectrograms must be generated after each iteration step. In this approach [12], they investigate the way of applying an augmentation to the spectrogram, not to the waveform. Therefore this method can be applied during training stage, without affecting on speech of training.

SpecAugment changes the spectrogram by deforming it in the time direction, modifying the frequency channels. These augmentation techniques have been selected in order to make the network effective against deformations and frequency losses.

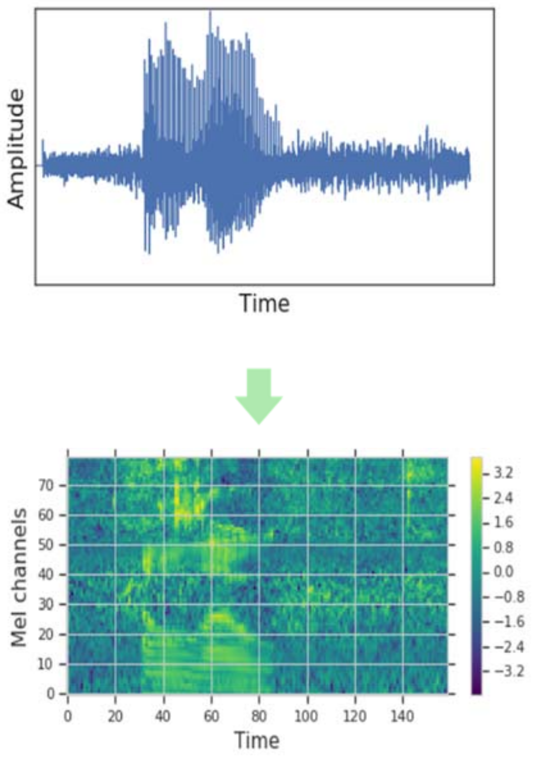


Figure 3. Visual representation of an audio waveform.

2.3. External Language Modeling

The prime job of the language model is to find out whether if particular sequence of words is appropriate or not in some context. It often used in fields like machine translation, handwriting recognition [13], spelling correction [14], augmentative communication [15] and Natural Language Processing tasks (natural language generation, word similarity) [16, 17, 18]. For speech recognition, to improve the accuracy and overall performance of your model independently trained language can be applied.

Recent researchers in Natural Language processing area have been using neural probabilistic language models instead of rule-based language models, because it can handle dataset with a large vocabulary, whereas rule-based language models very prone to errors when it comes to large datasets.

A quite big amount of word sequences are required to create the language models. Therefore, language model should be able to assign probabilities not only for small amounts of words, but also for the whole sentence. Nowadays there are a lot of preprocessed large dataset corpuses for creating language models. Even if there is a lack of data for some languages, it is possible to create datasets for language model, since the text data is easily available. Preprocessing step only requires removing words, that don't influence the whole context at all. Dataset can be collected from various sources like news portals, articles, books, which are mostly close to everyday used words.

For this particular ASR model we have collected the dataset from famous Kazakh books like "Abay zholy", "Kara sozder" etc. After collecting the raw text from these sources, we have applied simple preprocessing step, removing and replacing most of the unnecessary characters with space, removing all the punctuations, normalizing all words with lower case. We have transformed the sequence of words to a sequence of tokens. After that, we organized list of tokens into sequence of 50 words and 1 output, which gives us 51 words in 1 sequence. Overall, we have gotten 200 000 sequences. Later we have encoded each word with a unique number, because we have used an embedding layer, which expects input represented as an integers. To do the encoding, we have used Tokenizer class in the Keras API. Later, when we make predictions we can look up predicted number's associated word.

We have tried to train two similar models. To train these models we have used google colaboratory and overall training time was 7 hours. Performance of each model displayed in Figure 4. First model has of following structure:

1) Embedding layer with the vocabulary size 50.
2) LSTM layer with 50 neurons with L2 regularization.
3) Dropout regularization technique with 0.5 value.
4) LSTM layer with 50 neurons with L2 regularization.
5) Dropout regularization technique with 0.5 value.
6) Dense layer with 100 neurons and relu activation function for feature extraction.
7) Output layer which predicts the next word as a single vector.

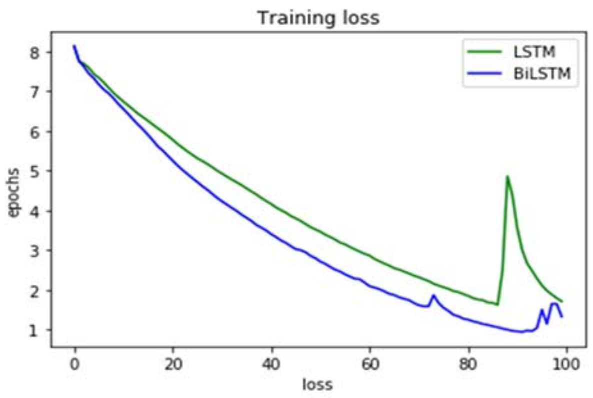


Figure 3. LSTM and BiLSTM models comparison

Second model has a BiLSTM model instead of LSTM:

1) Embedding layer with the vocabulary size 50.
2) BiLSTM layer with 50 neurons with L2 regularization.
3) Dropout regularization technique with 0.5 value.
4) BiLSTM layer with 50 neurons with L2 regularization.
5) Dropout regularization technique with 0.5 value.
6) Dense layer with 100 neurons and relu activation function for feature extraction.
7) Output layer which predicts the next word as a single vector.

Comparing the performance of two different models, we have chosen the second model with BiLSTM layer, since it showed very promising results comparing to model with LSTM layer in Table 1. Figure 4 demonstrates that LSTM model at some point started drastically increasing the loss. This unusual behavior can lead to problems like overfit, which is not observed in BiLSTM model.

Table 1. LSTM and BiLSTM performance comparison

Name	Loss	Accuracy	Epochs
LSTM	1.70	59.43	100
BiLSTM	1.32	73.71	100

3. CONNECTIONIST TEMPORAL CLASSIFIER

The CTC algorithm considers the order of the output labels of RNNs with ignoring the alignments by introducing a blank label, b. If the target labels are defined as L, then L' is an extended version of L with an extra blank symbol. The sequences over L' is defined as π, where $\pi \in L'T$, in which T represents the size if an input. The output sequence is labeled as z, where $z \in L \leq T$. It has a function F that maps the input sequence to output sequence $z = F(\pi)$. Basically, path π with its T can be mapped to a much shorter sequence z, by combining consecutive symbols into one symbol and by removing blank symbols. By having a input sequence and corresponding targets our RNN (recurrent neural network) can train and learn sequence mapping.

The CTC uses the forward-backward algorithm to compute the gradient of the loss function, $L(x, z)$. z' is the sequence over L'. Then, the variables, α and β, are initialized by

$$\alpha(1,u) = \begin{cases} y_b^1 \text{ if u = 1} \\ y_{z_1}^1 \quad if\ u = 2 \\ 0 \quad otherwise \end{cases} ,$$

$$\beta(T,u) = \begin{cases} 1 \quad if\ u = |z'|, |z'| - 1 \\ 0 \quad otherwise \end{cases}$$

where y_k^t is the softmax output of the label $k \in L'$. The forward and backward propagation is done as

$$\alpha(t,u) = y_{z'_u}^t \sum_{i=f(u)}^{u} \alpha(t-1,i) ,$$

$$\beta(t,u) = \sum_{i=u}^{g(u)} \beta(t+1,i) y_{z'_i}^{t+i},$$

where

$$f(u) = \begin{cases} u-1 & if\ z'_u = b\ or\ z'_{u-2} = z'_u \\ u-2 & otherwise \end{cases}, \quad g(u) = \begin{cases} u+1 & if\ z'_u = b\ or\ z'_{u+2} = z'_u \\ u+2 & otherwise \end{cases}$$

with boundary conditions:

$$\alpha(t,0) = 0, \forall t,$$
$$\beta(t,|z'|+1) = 0, \forall t$$

then, the error gradient at time t, a_k^t, is computed as

$$\frac{\partial ℶ(x,y)}{\partial a_k^t} = y_k^t - \frac{1}{p\left(\frac{z}{x}\right)} \sum_{u \epsilon B(z,k)} \alpha(t,u)\beta(t,u),$$

where $B(z,k) = \{u: z'_u = k\}$ and $p(z|x) = \alpha(T, |z'| - 1)$.

3.1. Beam search

This searching algorithm goes through the network output creating beams, which have a corresponding score. Figure 5 shows the beam's evolution: starting with an empty beam, we add all possible characters (there are only "a" and "b" in our case) on first iteration and we keep the ones with the best scores. This process is repeated until the whole NN output is processed.

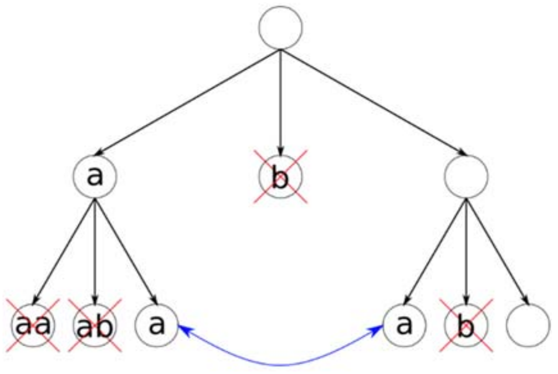


Figure 4. Beam seach

3.2. Optimization algorithms

Gradient descent [7] is well know and probably the most popular way to optimize deep neural networks. However, every well-known neural network frameworks contain implementations of various types of methods to optimize gradient descent.

Gradient descent is applied in order to minimize a cost function $J(\theta)$. It does this by updating every time the parameters in the negative direction of the function's gradient $\nabla_{\theta J} J(\theta)$. The learning rate η is a parameter that defines the step size in order to reach local minimum. Simply, we follow the direction of the slope downhill until we get to the minimum point.

Stochastic gradient descent

SGD in [8] does the same thing like simple gradient descent. However, it updates the parameters of the model taking original dataset's (m) part (mini-batch):

$$g = \frac{1}{m} \nabla_\theta \sum_i L\left(f\left(x^{(i)};\theta\right), y^{(i)}\right) ,$$

$$\theta = \theta - \epsilon_k \times g$$

Adagrad

This method simply makes an adaptive learning rate −η to depending on parameters. For example, infrequent parameters make a big update and frequent parameters make a small. Therefore, it is more suitable for working with rare data (sparse data).

In [11] Adagrad applies different learning rate at each time t for every parameter $\theta(i)$. Briefly, we set $g(t, i)$ as the gradient of loss function:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

Momentum

SGD has a hard time navigating areas where the curves a lot sharper in one depth comparing to another, which are around local optima. In these scenarios, SGD hesitates through the slopes while making slow progress through the bottom to the local minimum.

Momentum in [10] is a method that accelerates SGD in the appropriate direction. It is achieved by adding a fraction γ to the update vector.

$$\vartheta_t = \gamma\vartheta_{t-1} + \eta\nabla_\theta J(\theta)$$
$$\theta = \theta - \vartheta_t$$

The γ value is almost any case is equals to 0.9 or a somewhere around this value. Basically, when we are using momentum, we are pushing the ball down a hill. The momentum accumulates as the ball goes downhill, becoming faster. Identically, for parameter updates: The momentum increases if gradients go in the same direction and decreases updates if gradients directions change. As an outcome, we'll get more faster convergence with less oscillation.

Adam

Adam is the method that also computes the learning rate for the parameters that is adaptive [9]. Adam exponentially keeps decomposing average of past gradients M(t), like momentum:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)g_t^2$$

m_t and v_t are calculations of the first moment and the second moment of the gradients. When m_t and v_t are initialized as 0 vectors, it has been observed and stated that they are biased to zero, especially in the beginning of the time step and when the β_1 and β_2 are small.

$$\widehat{m_t} = \frac{m_t}{1-\beta_1^t}\ ,\ \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

These later used to update parameters:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\widehat{m_t}$$

The proposed values for β_1, β_2 and ϵ are 0.9, 0.999 and 10^{-8}, respectively.

4. Experiment

To train the CTC model, there has been used a python library Tensorflow. Actually, there are quite a lot of frameworks for building ASR systems like Kaldi, Sphinx. However, the advantage of Tensorflow is that, it allows us to use GPU of the computer and parallelize the tasks into tensors. As a speech data for a training a model, SDU corpus data has been used. It is a data, which has been collected in the base of university of Suleyman Demirel, which contains 70 native speakers with 360 sentences. It weighs about 5GB, but for illustration purposes, only part of it was used in this experiment.

The structure of the Neural Network is using a RNN (Recurrent Neural Network) with 100 hidden layers. Specifically, LSTM (Long-Short Term Memory) type of RNN has been used, for capturing all information in the sequence. As for optimization part, three algorithms have been considered (Adam, Momentum, Adagrad), which were discussed earlier. Each of them has advantages over the other two, but specifically for speech data, these algorithms were evaluated by LER (label error rate) and CTC loss.

Before feeding into neural network dataset has gone through data augmentation process with time warping technique. Number of epochs to train the model is 100. Output of the CTC model is actually sequence of characters including blanks (encoding part). Decoding part is done by using dynamic programming method, which is in this case greedy search. Greedy search was preferred instead of beam search, because it decodes faster, although beam search performs better. In order to avoid the overfitting problem we have taken the learning rate as 0.005. Network uses pre-trained language model while training and decoding in order to improve overall performance.

5. RESULTS AND DISCUSSIONS

After training the model three times with different optimization algorithms, we see the following outcomes (Table 2).

Table 2. Results of each optimizer

Name	Training cost	Validation cost	Training LER	Validation LER
Adagrad	166.774	242.164	0.970	0.980
Momentum	2.405	71.289	0.000	0.500
Adam	166.774	242.164	0.970	0.980

5.1. Adagrad

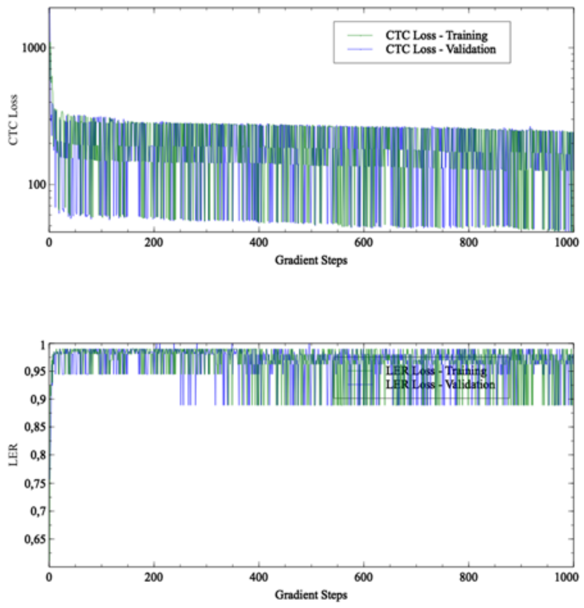


Figure 5. CTC loss and LER of Adagrad

After that, the loss value starts to hesitate drastically between 500 and 50 (approximately). At the same time, LER right from the beginning start to hesitate between 1 and 0.9(approximately), which does not allow the model to learn.

5.2. Momentum

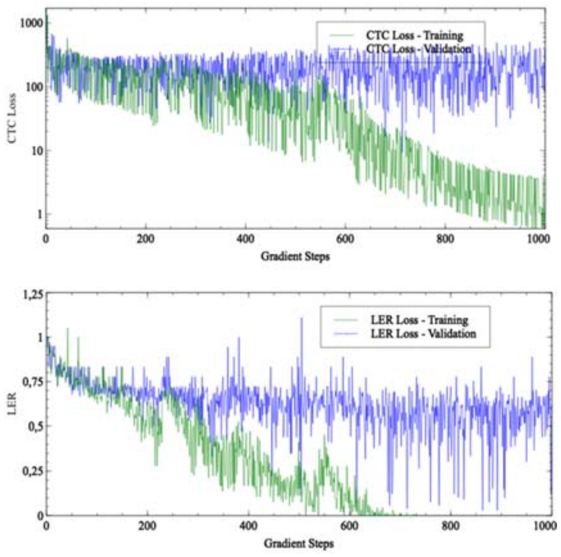
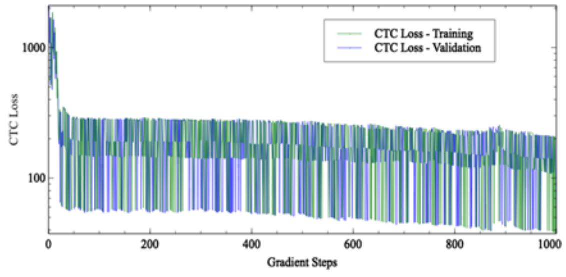


Figure 6. CTC loss and LER of Momentum

The visualization we see in Figure 7 shows that Momentum works a lot better than Adagrad. LER and CTC loss are continuously decreasing for training and validation sets. After the gradient steps reach the 900 epochs CTC loss goes down to almost 1, where LER shows the 0. This is may be prone to over fit because there is a quite big gap between training set and validation set. Applying other optimizing techniques like dropout and feeding even more data this problem can be avoided in the future.

Because of learning rate is equal to 0.005 decreasing process slows down little bit. Learning process is much better than Adagradoptimizer.

5.3. Adam



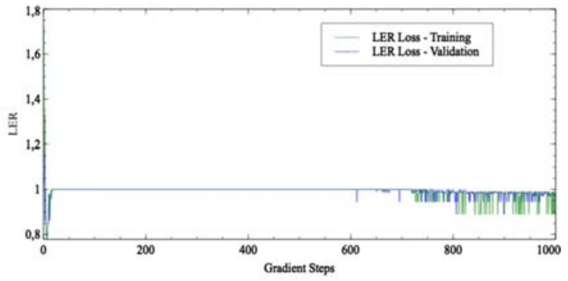


Figure 7. CTC loss and LER of Adam

As shown in Figure 5 CTC loss decreases at the beginning of the gradient steps and once again as in Adagrad optimizer starts to hesitate between two numbers with a big difference. LER on the other hand decreases to 1 after few iterations and after that does not change for a long gradient steps. Right after it reaches about 750 iteration LER starts to hesitate between 1 and 0.5(approximately), which is not a well performance.

CONCLUSION AND FUTURE WORKS

This paper shows a clear benefit of Momentum optimizer over Adam and Adagrad for CTC algorithm for speech recognition. The experiment showed that model with Momentum optimizer learns faster decreasing the CTC loss and LER after each gradient step, whereas Adagrad and Adam optimizers performed very poorly, showing a hesitation of errors from big number to small. Other than that, this paper shows the advanced algorithm called Connectionist Temporal Class for speech recognition in action. It also describes the clear benefits of this algorithm over the traditional method, which is HMM based model, which is the simplicity and effectiveness.

For the future work we are planning to add other regularization algorithms to make out CTC algorithm work a lot faster and accurate. To avoid the problems like overfit and exploding gradients, we are planning to add techniques like dropout, residual blocks and collect more data.

Funding: This work is supported by a grant from the Ministry of Education and Science of the Republic of Kazakhstan within the framework of the Project “AP05132648- Creating verbal and interactive robots based on advanced voice and mobile technologies”

Conflicts of Interest: The authors declare no conflict of interest.

REFERENCE LIST

[1] Mark G., Steve Y., The Application of Hidden Markov Models in Speech Recognition, *Foundations and Trends in Signal Processing,* Vol. 1, No. 3 (2007) 195–304

[2] Alex Graves, NavdeepJaitly. Towards End-to-End Speech Recognition with Recurrent Neural Networks, *Proceedings of the 31st International Conference on Machine Learning (2014),* Volume 32 Pages II-1764-II-1772

[3] Alex G., Santiago F., Faustino G., J¨urgen S., Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks, *Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006,* Pages 369-376

[4] YoshuaBengio, RéjeanDucharme Pascal Vincent, Christian Jauvin, A Neural Probabilistic Language Model, *Journal of Machine Learning Research 3 (2003) 1137–1155*

[5] Dong Yu, Jinyu Li, Recent Progresses in Deep Learning based Acoustic Models, *IEEE/CAA Journal of AutomaticaSinica(2017),* Volume 4, Pages 396 – 409

[6] Sam W., Alexander M. Rush. Sequence-to-Sequence Learning as Beam-Search Optimization, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing,* pages 1296–1306,November 1-5 (2016)

[7] Marcin Andrychowicz, MishaDenil, Sergio Gómez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, Nando de Freitas. Learning to learn by gradient descent by gradient descent, *30th Conference on Neural Information Processing Systems (NIPS 2016)*

[8] Leon B., Large-Scale Machine Learning with Stochastic Gradient Descent, *NEC Labs America, Princeton NJ 08542, USA*

[9] Diederik P. Kingma, Jimmy Lei Ba. ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION, *arXiv:1412.6980v9 [cs.LG] 30 Jan 2017*

[10] Nicolas L., Peter Richt´arik. Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods, *School of Mathematics, The University of Edinburgh (2017)*

[11] John Duchi, EladHazan, Yoram Singer. Adaptive Subgradient Methods for Online

Learning and Stochastic Optimization, *Journal of Machine Learning Research 12 (2011) 2121-2159*

[12] Daniel S. Park, William Chan, Yu Zhang, Chung-Cheng Chiu, BarretZoph, Ekin D. Cubuk, Quoc V. Le, SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition, *arXiv:1904.08779*

[13] Russell S. and Norvig P. Artificial Intelligence: A Modern Approach (2nd Ed.). *Pretice Hall. 2002.*

[14] Kukich Karen, Techniques for automatically correcting words in text. *ACM Computing Surveys.1992. 24(4), pp. 377-439.*

[15] Alan Newell, Stefan Langer, Marianne Hickey, The role of natural language processing in alternative and augmentative communication. *Natural Language Engineering. 1998. pp. 1-16*

[16] Kenneth Ward Church, A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text. *Second Conference on Applied Natural Language Processing. 1988. pp. 136–143*

[17] Peter F. Brown, John Cocke, Stephen Andrew Della Pietra, Vincent joseph DellaPietra, A Statistical Approach To Machine Translation, *Computational Linguistics, Volume 16, pp. 79-85*

[18] Jonathan J. Hull, Combining syntactic knowledge and visual text recognition: A hidden Markov model for part of speech tagging in a word recognition algorithm. *AAAI Symposium: Probabilistic Approaches to Natural Language. 1992. pp. 77–83.*