DOI: 10.35784/IAPGOS.234

COMPARISON OF OPTIMIZATION ALGORITHMS OF CONNECTIONIST TEMPORAL CLASSIFIER FOR SPEECH RECOGNITION SYSTEM

Yedilkhan Amirgaliyev[1,2], Kuanyshbay Darkhan[1,2], Aisultan Shoiynbek[2]
[1]Institute Information and Computational Technologies CS MES RK, Almaty, Kazakhstan, [2]Suleyman Demirel University, Almaty, Kazakhstan

Abstract. *This paper evaluates and compares the performances of three well-known optimization algorithms (Adagrad, Adam, Momentum) for faster training the neural network of CTC algorithm for speech recognition. For CTC algorithms recurrent neural network has been used, specifically Long-Short-Term memory. LSTM is effective and often used model. Data has been downloaded from VCTK corpus of Edinburgh University. The results of optimization algorithms have been evaluated by the Label error rate and CTC loss.*

Keywords: recurrent neural network, search methods, acoustic, systems modeling language

PORÓWNANIE ALGORYTMÓW OPTYMALIZACJI KLASYFIKATORA CZASOWEGO DO SYSTEMU ROZPOZNAWANIA MOWY

Streszczenie. *W artykule dokonano oceny i porównania wydajności trzech znanych algorytmów optymalizacyjnych (Adagrad, Adam, Momentum) w celu przyspieszenia treningu sieci neuronowej algorytmu CTC do rozpoznawania mowy. Dla algorytmów CTC wykorzystano rekurencyjną sieć neuronową, w szczególności LSTM, która jest efektywnym i często używanym modelem. Dane zostały pobrane z wydziału VCTK Uniwersytetu w Edynburgu. Wyniki algorytmów optymalizacyjnych zostały ocenione na podstawie wskaźników Label error i CTC loss.*

Słowa kluczowe: rekurencyjna sieć neuronowa, metody wyszukiwania, akustyka, język modelowania systemów

Introduction

There have been many techniques for recognizing speech and variety of tasks like voice pattern recognition, which helps to identify human by his voice [12]. Considering the fact that speech data is complicated in terms of segmentation that is [1], 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 [7], which involves other pre-trained models like acoustic model, language model etc. However, recent researches have shown that by using recurrent neural networks [9], 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) [7] would require data for training language model and acoustic model. This advanced algorithm called Connectionist-Temporal-Classifier [8], 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 overfit or underfit, especially with speech data. Labelling an unsegmented data is very common and often difficult problem in the sequence-to-sequence models. Straightforward way to solve this problem 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 [3], which gives the phoneme representation of the given speech (Fig. 1).

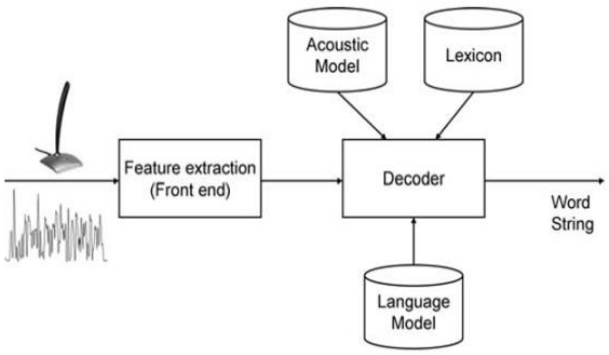


Fig. 1. Traditional ASR system

Labelling an unsegmented data is very common and often difficult problem in the sequence-to-sequence models. Straightforward way to solve this problem 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 [3], which gives the phoneme representation of the given speech (Fig. 1).

Connectionist temporal classifier [8] require 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 [13]. 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).

This paper organized as follows. Section 2 contains the information about CTC algorithm, Beam search and optimization algorithms, which will be considered in the experiment. Section 3 contains the experiment itself, which is about building a neural network, used optimization algorithms and dataset. Section 4 illustrates the outcomes of the experiment that shows a result of optimization algorithms comparing with each other (Adagrad, Adam, and Momentum). Section 5 concludes the whole experiment by choosing the best optimizer for CTC algorithm.

1. Encoder and 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. $P(y_1, \ldots, y_O|x_1, \ldots, x_T)$ 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 y_o, 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.

$$P(y_1,\ldots,y_O|x_1,\ldots,x_T) = \prod_{o=1}^{O} P(y_o|y_1,\ldots,y_{o-1},c_O)$$

Since the probability $P(y_{1,\ldots,} y_O | x_{1,\ldots,} x_T)$ $P(y_{1,\ldots,}y_O|x_{1,\ldots,}x_T)$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 y_o be a vector representation of the output symbol yo, where y_o 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 y_o is computed as

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

where so denotes the output of a recurrent hidden layer f(·) with inputs y_{o-1}, s_{o-1}, and c_o. $g(\cdot)$ 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 y_{o-1} is also fed to the softmax function g(·) to capture the bigram dependency between consecutive words [3]. We have also investigated a simpler output function without the dependence on the previous output y_{o-1}, i.e. $P(y_o|y_1, \ldots, y_{o-1}, c_o) = g(s_o, c_o)$.

Encoder

As discussed above, the computation of the conditional probability relies on the availability of the context vector c_0 for each output y_0. 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) [8]:

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

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 c_0 is global, for instance, $c_0 = hT$. 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 [14]. 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 α_{ot} 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.

2. 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*. For the set of target labels, *L*, and its extended set with the additional CTC blank label, $L' = L \cup \{b\}$, the path, π, is defined as a sequence over L', that is, $\pi \in L'^T$, where *T* is the length of the input sequence, *x*. Then, the output sequence, $z \in L^{\leq T}$, is represented by $z = F(\pi)$ with the sequence to sequence mapping function *F*. *F* maps any path π with the length *T* into the shorter sequence of the label, *z*, by first merging the consecutive same labels into one and then removing the blank labels. Therefore, any sequence of the raw RNN outputs with the length *T* can be decoded into the shorter labelling sequence, *z*, with ignoring timings. This enables the RNNs to learn the sequence mapping, $z = G(x)$, where *x* is the input sequence and *z* is the corresponding target labelling for all (x, z) in the training set, *S*. More specifically, the gradient of the loss function $L(x, z) = -\ln p(z/x)$ is computed and fed to the RNN through the softmax layer, of which the size is $|L'|$.

The CTC algorithm employs the forward-backward algorithm for computing the gradient of the loss function, $L(x, z)$. Let z' be the sequence over L' with the length of $2|z|+1$ where $z'_u = b$ for odd u and $z'_u = z_{u/2}$ for even u. Then, the forward variable, α, and the backward variable, β, are initialized by

$$\alpha(1, u) = \begin{cases} y^1_b & if\ u = 1 \\ y^1_{z_1} & if\ u = 2 \\ 0 & otherwise \end{cases}$$
$$\beta(T, u) = \begin{cases} 1 & if\ u = |z'|, |z'|-1 \\ 0 & otherwise \end{cases}$$

where y^t_k is the soft max output of the label $k \in L'$ at time *t*. The variables are forward and backward propagated as

$$\alpha(t, u) = y^t_{z'_u} \sum_{i=f(u)}^{u} \alpha(t-1, i),$$
$$\beta(t, u) = \sum_{i=u}^{g(u)} \beta(t+1, i) y^{t+i}_{z'_i},$$

where

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

With boundary conditions:

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

Then, the error gradient with respect to the input of the softmax layer at time *t*, $a^t{}_k$ t, a^t_kt, a^t_k, is computed as

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

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

Beam search

This algorithm iterates through the NN output and creates text candidates (called beams) which are scored. Figure 2 shows an illustration of the evolution of beams: we start with the empty beam, then add all possible characters (we only have "a" and "b" in this example) to it in the first iteration and only keep the best scoring ones. The beam width controls the number of surviving beams. This is repeated until the complete NN output it processed.

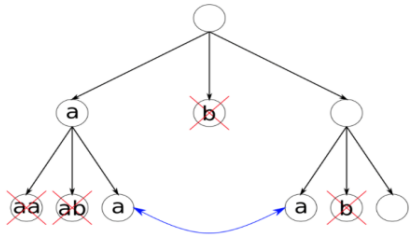


Fig. 2. Beam search

3. Optimization algorithms

Gradient descent [14] is one of the most popular algorithms to perform optimization and by far the most common way to optimize neural networks. At the same time, every state-of-the-art Deep Learning library contains implementations of various algorithms to optimize gradient descent. These algorithms, however, often used as black-box optimizers, as practical explanations of their strengths and weaknesses are hard to come by.

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \epsilon R^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_{\theta J}(\theta)$ w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.

Stochastic Gradient Descent

SGD in [10] updates model parameters θ in the negative direction of the gradient (g) by taking a subset or a mini-batch of data of size (m):

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

Adagrad

This method simply allows the learning Rate -η -η to adapt based on the parameters. Therefore, it makes big updates for infrequent parameters and small updates for frequent parameters. For this reason, it is well suited for dealing with sparse data.

Adagrad uses a different learning rate in [5] for every parameter *θ(i)* θ(i) at every time step *t* t, we first show Adagrad's per-parameter update, which we then vectorise. Briefly, we set *g(t, i)* g(t, i) to be the gradient of the loss function w.r.t. to the parameter *θ(i)* θ(i) θ(i) at time step ***t*t.**

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

Momentum

SGD has trouble navigating ravines, i.e. areas where the surface curves much more steeply in one dimension than in another, which are common around local optima. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum.

Momentum in [2] is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ γof the update vector of the past time step to the current update vector.

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

The momentum term γ is usually set to 0.9 or a similar value. Essentially, when using momentum, we push a ball down a hill. The ball accumulates momentum as it rolls downhill, becoming faster and faster on the way (until it reaches its terminal velocity, if there is air resistance, i.e. γ < 1). The same thing happens to our parameter updates: The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, we gain faster convergence and reduced oscillation.

Adam

Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter [15]. In addition to storing an exponentially decaying average of past squared gradients, Adam also keeps an exponentially decaying average of past gradients *M (t)*M(t), similar to 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 estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients respectively, hence the name of the method. As m_t and v_t are initialized as vectors of 0's, the authors of Adam observe that they are biased towards zero, especially during the initial time steps, and especially when the decay rates are small (i.e. β_1 and β_2 are close to 1).

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

They then use these to update the parameters:

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

The authors propose default values of 0.9 for β_1, 0.999 for β_2, and 10^{-8} for ϵ. They show empirically that Adam works well in practice and compares favourably to other adaptive learning-method algorithms.

4. Results

After training the model three times with different optimization algorithms, we see the following outcomes (Table 1):

Table 1. Result 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

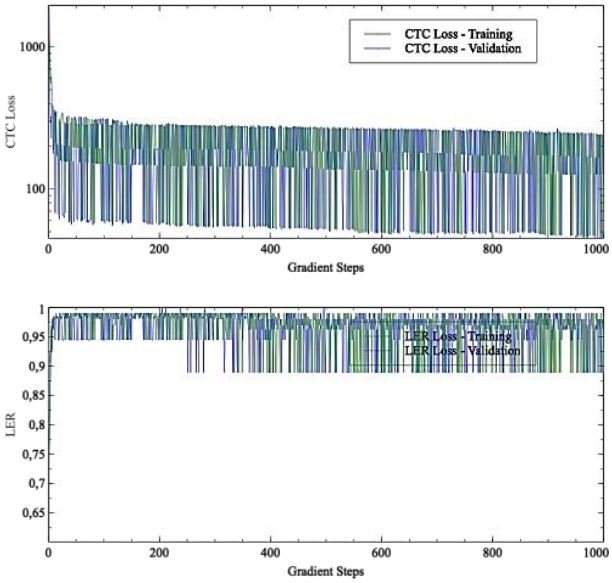


Fig. 3. CTC loss and LER of Adagrad

As we can see in Figure 3 CTC loss at the beginning of each iteration starts to decrease as it should, but 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.

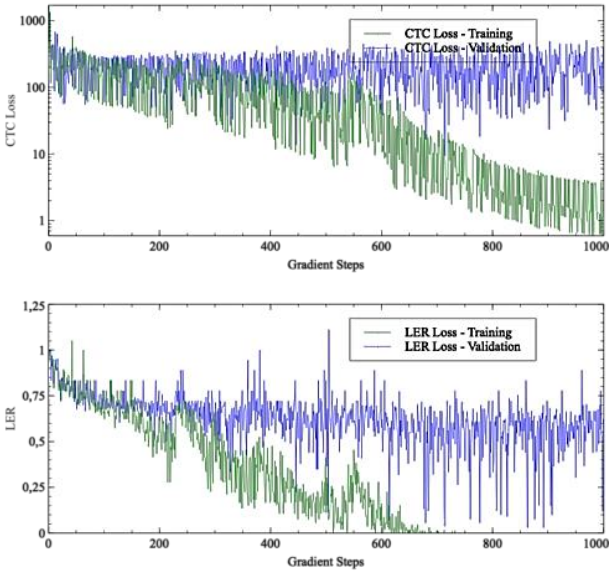


Fig. 4. CTC loss and LER of Momentum

The visualization we see in Figure 4 shows that Momentum works a lot better than Adagrad. LER and CTC loss are continuously decreasing for training and validation sets. Because of learning rate is equal to 0.005 decreasing process slows down little bit. Other than that, learning process is doing well.

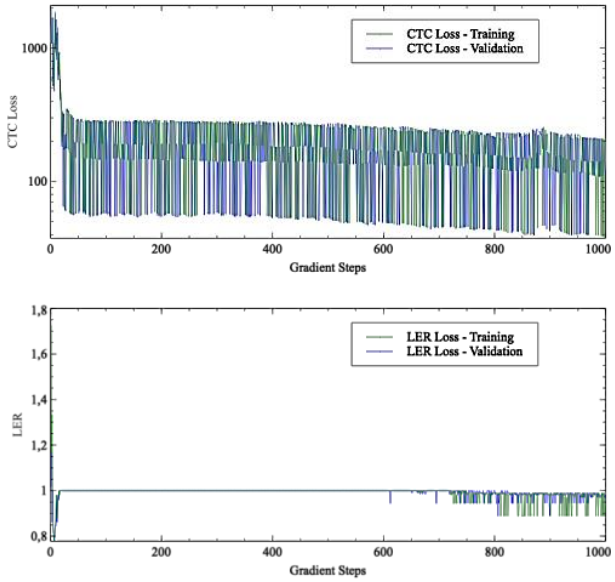


Fig. 5. CTC loss and LER of Adam

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.

5. Conclusion

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.

References

[1] Amirgaliev Y., Hahn M., Mussabayev T.: The speech signal segmentation algorithm using pitch synchronous analysis. Journal Open Computer Science 7(1)/2017, 1–8.
[2] Andrychowicz M., Denil M., Colmenarejo S.G., Hoffman M.W., Pfau D., Schaul T., Shillingford B., de Freitas N.: Learning to learn by gradient descent by gradient descent. 30th Conference on Neural Information Processing Systems NIPS 2016.
[3] Bahdanau D., Cho K., Bengio Y.: Neural machine translation by jointly learning to align and translate. Proc. ICLR, 2015.
[4] Bengio Y., Ducharme R., Vincent P., Jauvin C.: A Neural Probabilistic Language Model. Journal of Machine Learning Research 3/2003, 1137–1155.
[5] Bottou L.: Large-Scale Machine Learning with Stochastic Gradient Descent. NEC Labs America, Princeton.
[6] Duchi J., Hazan E., Singer Y.: Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. Journal of Machine Learning Research 12/2011, 2121–2159.
[7] Gales M., Young S.: The Application of Hidden Markov Models in Speech Recognition. Foundations and Trends in Signal Processing 1(3)/2007, 195–304.
[8] Graves A., Fernandez S., Gomez F., Schmidhuber J.: Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA, 2006.
[9] Graves A., Jaitly N.: Towards End-to-End Speech Recognition with Recurrent Neural Networks. Proceedings of the 31st International Conference on Machine Learning 2014.
[10] Kingma D.P., Ba J.: Adam: A Method For Stochastic Optimization. Proc. 3rd International Conference for Learning Representations. 2015 arXiv:1412.6980v9.
[11] Loizou N., Richtarik P.: Momentum and Stochastic Momentum for Stochastic Gradient, Newton, Proximal Point and Subspace Descent Methods. 2017, arXiv:1712.09677v2
[12] Mussabayev R.R., Amirgaliyev N., Tairova A.T., Mussabayev T.R., Koibagarov K.C.: The technology for the automatic formation of the personal digital voice pattern. Application of Information and Communication Technologies AICT 2016.
[13] Schuster M., Paliwal K.K.: Bidirectional recurrent neural networks. Signal Processing. IEEE Transactions 45(11)/1997, 2673–2681.
[14] Sutskever I., Vinyals O., Le Q.V.: Sequence to sequence learning with neural networks. Advances in Neural Information Processing Systems 2014, 3104–3112.
[15] Wiseman S., Rush A.M.: Sequence-to-Sequence Learning as Beam-Search Optimization. Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing 2016.
[16] Yu D., Li J.: Recent Progresses in Deep Learning based Acoustic Models. Tencent AI Lab, Microsoft AI and Research, 2018.

Ph.D. Yedilkhan Amirgaliyev
e-mail: amir_ed@mail.ru

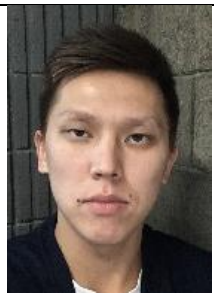
Doctor of technical sciences, professor, head of the laboratory of mathematical and computer modelling of the Institute of Information and computing technologies of the Science Committee of RK MES. The Institute is the leading organization in the field of information technology in the country. The main directions of the research laboratory is an intelligent decision-making systems, robotics, wireless sensor technology, computer modelling of technological processes, etc.

ORCID ID: 0000-0002-6528-0619

M.Sc. Kuanyshbay Darkhan
e-mail: darkhan.kuanyshbay@sdu.edu.kz

Ph.D. student on a specialty "Information Systems", a programmer at the Institute of Institute of Information and Computational Technologies of the Ministry of Education and Science CS of the Republic of Kazakhstan. Research interests: mathematical modelling of discrete systems, evacuation tasks, operations research, technology design of complex systems.

ORCID ID: 0000-0001-5952-8609



M.Sc. Aisultan Shoiynbek
e-mail: aisultan.shoiynbek@sdu.edu.kz

Ph.D. student on a specialty "Information Systems", a programmer at the Institute of Institute of Information and Computational Technologies of the Ministry of Education and Science CS of the Republic of Kazakhstan. Research interests: mathematical modelling of discrete systems, evacuation tasks, operations research, technology design of complex systems.

ORCID ID: 0000-0002-9328-8300

otrzymano/received: 30.05.2019 *przyjęto do druku/accepted: 15.06.2019*