Ministry of Education and Science of the Republic of Kazakhstan
Suleyman Demirel University

YERSULTAN ABDIKHALIYEV

REALIZATION OF ALGORITHM OF SELF-GENERATING NEURAL NETWORKS

THESIS

Presented in Partial Fulfillment for the
Department of Science
(degree code: 6M060100)
Department of Mathematics and Natural Sciences
Faculty of Engineering and Natural Sciences

Supervisor: **KAYSAR TULENBAYEV**

Kaskelen, 2019

Abstract

The development of various spheres of human activity is associated with the generation and accumulation of a huge amount of data that can contain the most important practical information. However, significant benefit from this information can be extracted only with proper processing and analysis of this data. Recently, there has been an increased interest in the field of artificial intelligence, and methods of automating the extraction of knowledge based on data mining are actively developing. Self-generating neural networks are built on the principle of biological, of course, with a number of assumptions, they have a huge number of simple processes with many connections. Like the human brain, these networks are capable of learning. Self-generating neural networks find their application in areas such as computer vision, speech recognition, processing of natural language, etc. The thesis provides an analysis of the development of the theory of neural networks, their classification and mathematical formulation of the task of recognizing pattern recognition.

Keywords: neural network, machine learning, sentiment analysis, self-generating neural network, deep learning, natural language processing.

Аңдатпа

Адам қызметінің әртүрлі салада дамуы көптеген маңызды практикалық ақпараттарды, өзінде қамтитын үлкен көлемді деректерді жинақтаумен байланысты. Алайда, осы деректерді дұрыс анализ жасағанда ғана бұл ақпараттардан айтарлықтай пайда алуға болады. Соңғы жылдары жасанды интеллект саласына қызығушылықтың өсуі байқалуда және деректерді интеллектуалды өңдеу негізінде білім алу әдісі қарқынды дамуда. Өздігінен генерациялайтын нейрондық желілер, алуан түрлі байланысы бар көптеген қарапайым процесстерге ие, биологиялық қағидаларға негізделген. Адам миы секілді бұл желілер білім алуға (үйренуге) қабілетті. Өзін-өзі дамытатын нейрондық желілер компьютерлік көру, сөйлеуді тану, табиғи тілдерді өңдеу және т.б. салаларында қолдануды табады. Ғылыми жұмыста нейрондық желілердің даму теориясына анализ жасалды, сонымен қатар олардың жіктелуі, үлгіні тану әдісінің математикалық тұжырымдамасы қарастырылды.

Түйін сөздер: нейрондық желі, машинамен оқыту, мәтін талдау, өзін-өзі дамытатын нейрондық желі, терең білім, табиғи тілдерді өңдеу.

Аннотация

Развитие различных сфер человеческой деятельности связано с генерацией и накоплением огромного количества данных, которые могут содержать в себе важнейшую практическую информацию. Однако значительную пользу из этой информации можно извлечь лишь при правильной обработке и анализе этих данных. В последнее время наблюдается повышенный интерес к области искусственного интеллекта, и активно развиваются методы автоматизации извлечения знаний на основе интеллектуального анализа данных. Самогенерирующие нейронные сети построены по принципу биологических, конечно, с рядом допущений, в них действует огромное количество простых процессов со множеством связей. Подобно человеческому мозгу эти сети способны обучаться. Самогенерирующие нейронные сети находят свое применение в таких областях как компьютерное зрение, распознавание речи, обработка естественного языка и т.д. В работе приводится анализ развития теории нейронных сетей, их классификация и математическая постановка задачи обучения распознаванию образов.

Ключевые слова: нейронная сеть, машинное обучение, анализ тональности, самогенерирующая нейронная сеть, глубокое обучение, обработка естественного языка.

Acknowledgements

I would first like to thank my thesis advisor PhD Tulenbayev K. of the Math Department at Institute of Math and Math Modeling. The door to Prof. Tulenbayev office was always open whenever I ran into a trouble spot or had a question about my research or writing. He consistently allowed this paper to be my own work, but steered me in the right the direction whenever he thought I needed it.

I would also like to thank the external reviewer Dr Kavokin A. of the Math Department at Institute of Math and Math Modeling, who was involved in the validation survey for this research project. Without his passionate participation and input, the validation survey could not have been successfully conducted.

I would also like to acknowledge PhD Kadyrov Sh. of the Natural Sciences Department at Suleyman Demirel University as the second reader of this thesis, and I am gratefully indebted for his very valuable comments on this thesis.

Finally, I must express my very profound gratitude to my parents and to my future spouse for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you.

Abbreviations

AdaDelta – Adaptive Learning Rate Method
AdaGrad - Adaptive gradient algorithm
Adam - Adaptive Moment Estimation
ANN - Artificial Neural Network
LSTM - Long Short-Term Memory
NAG – Nesterov Accelerated Gradient
NN - Neural Network
NLP - Natural Language Processing
RNN - Recurrent Neural Network
SGNN - Self-Generating Neural Network
SVM - Support Vector Machines
VSM - Vector Space Models

To my family

Оглавление

1. Introduction

Thesis on *Self-Generating Neural Networks* is related to the fact that the method of processing information by the human brain is fundamentally different from the methods used by conventional digital computers. The brain is an extremely complex, non-linear, parallel computer. It has the ability to organize its structural components, called neurons, so that they can perform specific tasks (such as pattern recognition, processing signals of the senses, motor functions) many times faster than the fastest computers can afford. The brain has a perfect structure that allows you to build your own rules based on experience. Experience accumulates over time.

The concept of the development of neurons in the brain is associated with the concept of plasticity of the brain – the ability to tune the nervous system in accordance with the environment. Similarly, in *Self-Generating Neural Networks*, artificial neurons are adjusted and the structure of the neural network is formed. In general, the neural network *Self-Generating Neural Networks* is a machine that simulates how the brain processes a particular task. This network is usually implemented using electronic components or simulated by the program.

One of the areas of artificial intelligence Natural Language Processing, allows computing systems to extract various information from natural language.

An important task of *Natural Language Processing* is the task of determining the *Tonality of the Text (Sentiment Analysis)*. The task of determining the tone of the text is to extract the author's emotional evaluation, expressed in the text.

The *Sentiment Analysis* can be applied in various fields, for example, in sociology (to determine the attitude of users of social networks to certain events), marketing to determine the ratio of customers to certain products, psychology (to determine depression among users of social networks).

There are a large number of works devoted to the processing of natural

language and, in particular, the *Sentiment Analysis*. But most of them are adapted to apply to the English language. This paper deals with the task of classifying a Kazakh language text.

In this thesis, an accessible data set was used for Kazakh texts [5], with the addition of my work and library (in our case approximately 20015 reviews, of which 7002 are positive, 6945 are negative and 6068 are neutral). Evaluation of posts was carried out on a scale from -1 to 1.

The difficulty of analyzing the tonality lies in the presence of an emotionally enriched language – slang, polysemy, uncertainty, sarcasm, all these factors are misleading not only people, but also computers.

Before any data processing requires preprocessing (data cleaning). In this stage, all html tags, punctuation, symbols are removed. This operation is performed using the python library - "Beautiful Soup". Also, all numbers and links in the text are replaced by tags. Further in the text there are so-called "stopwords"– these are private words in the language, which basically do not carry any meaningful meaning (for example, in English these are words like "the "at "about etc.). Stopwords are removed using the *Python Natural Language Toolkit (NLTK)* package. As a consequence, a consistent step is the conversion of words into vectors. The fact is that the computer, as well as mathematical formulas, is easier to work with numbers, and not a set of words. Therefore, we need to present any text as a vector of numbers. To do this, you can make a dictionary with all the words, i.e. merge all the words found in the texts into one large dictionary and replace the words from the text with the index in the dictionary. There are several types of word embedding methods, including *word2vec* and *fastText*, which are most popular for representing words. This study uses simple tokenization in neural networks and the most popular *word2vec* word embedding algorithm. Finally, Self-Generating Neural Network architectures are applied to the learning model, then they are saved and used to predict new text to test performance.

The programming language Python was used to implement proposed models in this work since it has a large number of scientific libraries for data processing and machine learning algorithms. To perform machine learning or deep learning tasks, there exist several tools or libraries such as *TensorFlow*, *Theano*, *Scikit-learn* , *Keras* and etc.

2. Neural Network

2.1 Biological Neural Network

The neural network contains nodes, analogs of nerve cells, neurons (neural-like elements) and their connections (synaptic connections). A model of a neuron in interaction with other neurons of a neural network is shown in 2.1.

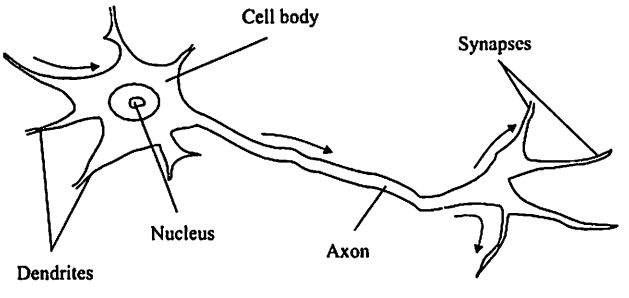


Рис. 2.1: Biological Neural Network

Here V_{ki} is the energy fraction of pulses V_k produced by other neurons and received by the dendrites of neuron i; W_{ki} – dendrite weights; h_i – thresholds. In turn, the generated impulse V_i also spreads through the dendrites of neurons with which neuron i is connected with the help of a branching axon. In accordance with the law of energy distribution, the quantity V_i is divided in proportion to the weights of the dendrites of the "receiving"neurons (in practical models, this energy distribution is usually not implemented). Each neuron is controlled either from the outside, or according to the principle of self-control using feedback. You can adjust the values of the weights of synaptic connections W_i and the values of thresholds h_i. Such regulation, implemented in different models, determines the possibility of learning and self-learning of the network. It sets the paths of signals passing through the network in the simplest way forming links.

The artificial neural networks (ANN) are extremely simplified analogues of natural neural networks. The nervous systems of animals and humans are much

more complex than the devices that can be created using modern technology. However, for successful solution of many practical problems, it turned out to be quite enough to "look"only at the general principles of the functioning of the nervous system. Some varieties of ANN are mathematical models that have only a remote resemblance to neurophysiology, which does not interfere with their practical application. ANN in 2.2 allows us to submit the following:

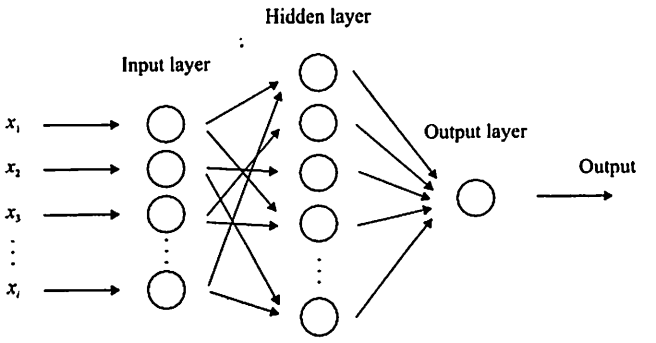


Рис. 2.2: Artificial Neural Network

Functions $f(x)$ are different. In the simplest case, it coincides with the linear form – the specified argument, i.e., over all dendrites, taking their weights into account, they are summed and compared with the threshold (w_k – input layer, V_k – hidden layer).

$$f(x) = \sigma(w_i V_j + w_j V_i), \quad \text{where} \quad \sigma(x) = \begin{cases} 0, & if\ x < 0 \\ x, & if\ x \geq 0 \end{cases} \tag{2.1}$$

The threshold value is the magnitude of the neuron signal or determines the magnitude of the signal magnitude. The network recognizes the input (receptor) layer, which perceives external signals, and the output sing, which determines the solution of the problem. Neurons operate with numbers in the range [0,1] or [-1,1]. The network operates in two modes: *learning mode* and *recognition mode.*

We will introduce classification and we will distinguish problems by conditional complexity (simple, medium and high complexity) and by whether we know how to solve it:

- Problems of low/medium complexity, which are exactly known how to solve:
  - Solve a simple equation;
  - Print the document on the printer;

- Display the program window etc.

Such problems are solved with the help of ordinary computer programs. Nothing complicated and unusual.

- Problems of low/medium complexity, which are partially known how to solve:
  - Simple prediction;
  - Approximate solution of equations;
  - Calculation of errors etc.

Such problems can be solved with the help of special statistical methods.

- Problems of high complexity, which is not clear how to solve:
  - Pattern recognition;
  - Speech recognition;
  - Complex forecasts.

Such tasks are solved using the ANN.

When solving problems using a neural network, a standard neural network configuration is selected, but given the complexity and peculiarities of the task, the selection of existing configurations can be problematic. If the task cannot be reduced to any of the known types of neural network, it is necessary to solve the complex problem of the synthesis of a new configuration. To determine the structure of the neural network model, it is necessary to solve several problems: to construct a classification of neural networks; analyze existing neural networks; to develop the main criteria for the selection of neural networks for model building; identify the main characteristics to determine the quality of the model based on the neural network.

2.2 Perceptron

Perceptron is a mathematical model with the construction of a brain model. By "brain model"is meant any theoretical system that seeks to explain the physiological

functions of the brain using the known laws of physics and mathematics, as well as the well-known facts of neuroanatomy and neurophysiology. A perceptron (the strict definition of which will be given below) is a transmission network consisting of three types of signal generators:

- Sensory elements;
- Associative elements;
- Reactive elements.

The generating functions of these elements depend on signals arising either somewhere inside the transmission network or, for external elements, on signals coming from the external environment. Thus, perceptrons will allow you to create a set of "associations"between input stimuli and the necessary response at the output. In biological terms, this corresponds to the transformation, for example, of visual information into a physiological response from motor neurons. According to modern terminology, perceptrons are classified and shared as ANN in such sections:

- With one hidden layer;
- Direct propagation;
- With threshold transfer function.

Perceptron consists of three types of elements: S, A and R. S – elements are a layer of receptors. These receptors are connected to A-elements, which are adders with a threshold (that is, formal neurons). The signals from the A-elements are transmitted to the adder R, and the signal from the $i-th$ associative element is transmitted with the coefficient k_i. And A or R – elements (which are threshold) calculate some linear form (usually the sum of weight coefficients) from their inputs and compare it with a given value – the threshold. If an A-element has n inputs, then it must specify n weights $w_1, w_2, \ldots, w_n$ and a threshold θ. The perceptron yields 1 if the linear form from inputs with coefficients w_i exceeds and -1 otherwise.

$$f(x) = sign\left(\theta + \sum_{i=1}^{n} w_i x_i\right) \tag{2.2}$$

The system of connections between receptors S and A – elements, as well as the thresholds of A-elements are selected in some random, but fixed manner, and learning consists only in changing the coefficients w_i. We teach the perceptron to separate two classes of objects, and demand that the output of the perceptron should be positive when presenting objects of the first class, and negative when they are presented to objects of the second class. The initial coefficients w_i are assumed to be zero. Next, we present a training sample: objects (for example, circles or squares) indicating the class to which they belong. Show perceptron object first class. However, some A-elements will be excited. We increase the coefficients w_i corresponding to these excited elements by 1. Then we present the object of the second class and the coefficients w_i of those A-elements that will be excited during this show, decrease by 1. We will continue this process for the entire training set. As a result of the training, the values of the weights of the links w_i will be formed [1].

After training, the perceptron is ready to work in recognition mode or generalization In this mode, objects that are "not familiar"are presented to the perceptron, and the perceptron must determine which class they belong to. The work of the perceptron is as follows: upon presentation of the object, the excited A-elements transmit a signal to the R-element equal to the sum of the corresponding coefficients w_i. If this sum is positive, then it is decided that this object belongs to the first class, and if it is negative, then to the second.

$$f(x) = \sigma(\varepsilon) = \begin{cases} 1, & if \quad \varepsilon \geq 0 \\ 0, & if \quad \varepsilon < 0 \end{cases} \tag{2.3}$$

The structure of perceptron is shown in 2.3.

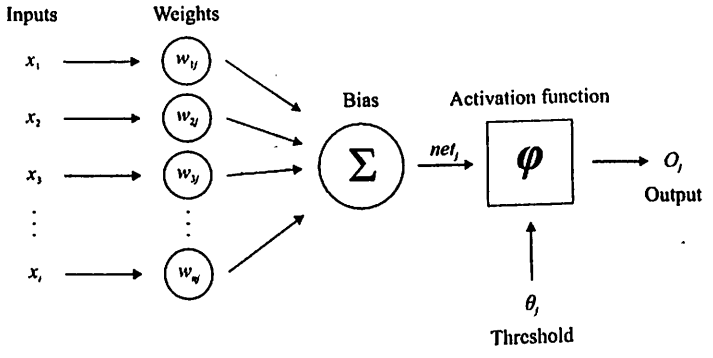


Рис. 2.3: Perceptron

Currently, the flow of information per person per day and received through

technical means is 27 mb per day, which is equivalent to 176 newspapers, of which only a small percentage can be considered useful information. Taking into account the fact that the capabilities of technical means of information processing, as well as specialized devices and programs designed to create content and deliver it to the consumer, double every 18-24 months, the chances of successfully mastering at least 10% of information for an ordinary person fall every year.

2.2.1 The Task of Exclusive OR

It is easy to build neurons that implement the logical functions AND, OR, NOT of the binary variables x_1 and x_2 (2.4).

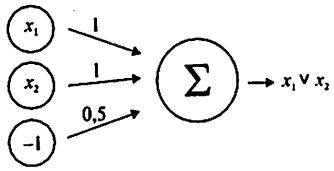


Рис. 2.4: 1

$$\begin{aligned} x_1 \vee x_2 &= \left[x_1 + x_2 - \tfrac{1}{2} > 0\right] \\ x_1 \wedge x_2 &= \left[x_1 + x_2 - \tfrac{3}{2} > 0\right] \\ \neg x_1 &= \left[x_1 + \tfrac{1}{2} > 0\right] \end{aligned} \tag{2.4}$$

However, the function $x_1 \oplus x_2 = [x_1 \neq x_2]$ is exclusive OR impossible in principle to be realized by one neuron with two inputs x_1 and x_2, since the sets of zeros and ones of this function are linearly inseparable.

Now we build a composition of several neurons. For example, an exclusive OR can be implemented by submitting the outputs of an AND-neuron and an OR-neuron to the input of another OR-neuron(2.5).

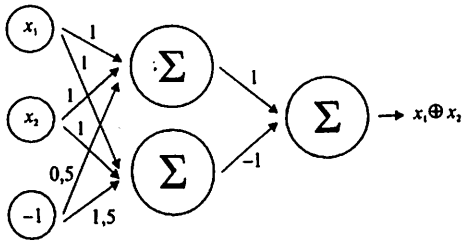


Рис. 2.5: 2

$$x_1 \oplus x_2 = (x_1 \vee x_2) - (x_1 \wedge x_2) - \frac{1}{2} > 0 \tag{2.5}$$

Further generalization of this idea leads to the construction of multilayer neural networks consisting of a huge number of connected neurons and resembling natural neural networks. The values of all n signs are simultaneously fed to the input of all N neurons of the first layer. Then their output values are fed to the input of all M neurons of the next layer. In this case, this layer is the output – this network is called a *two-layer network*. In general, the network may contain an arbitrary number of layers. All layers, except for the last, are called *hidden*.

2.3 Activation Functions

One of the most important aspects of neural networks is the activation function, which introduces nonlinearity into the network, making them universal function proxies.

The activation function is a way to normalize the input data. That is, if you have a large number at the entrance, by passing it through the activation function, you will get an output in the desired range.

If the activation function is designated as f, the output signal Y of the neuron depends on the vector of input values $X_1, X_2, \ldots, X_n$ and the vector of weight coefficients $W_1, W_2, \ldots, W_n$ as follows:

$$Y = f\left(X_1 \cdot W_1 + X_2 \cdot W_2 + \ldots\ldots + X_n \cdot W_n\right) \tag{2.6}$$

There are many activation functions. The most common ones are:

- *Linear transfer function* – the signal at the output of the neuron is linearly related to the weighted sum of the signals at its input.

$$f(x) = tx, \quad \text{where } t - \text{function parameter} \tag{2.7}$$

In ANN with a layered structure, neurons with transfer functions of this type, as a rule, constitute the input layer. In addition to a simple linear function, its modifications can be used. For example, a semilinear function (if its argument is less than zero, then it is equal to zero, and in other cases behaves as linear) or step function (linear function with saturation), which

can be expressed by the formula:

$$f(x) = \begin{cases} 0, & if\ x \leq 0 \\ 1, & if\ \ x \geq 1 \\ x, & else \end{cases} \tag{2.8}$$

- *Threshold transfer function* is a differential. Until the weighted signal at the input of the neuron reaches a certain level T – the signal at the output is zero. As soon as the signal at the input of the neuron exceeds the specified level, the output signal changes abruptly by one. The mathematical writing of this function looks like:

$$f(x) = \begin{cases} 1, & if\ x \geq T \\ 0, & else \end{cases} \quad \text{where} \quad T = -w_0 x_0 \quad \text{(shift function activation)} \tag{2.9}$$

  Accordingly, x should be understood as the weighted sum of signals at the inputs of the neuron, without taking into account this term. Due to the fact that this function is not differentiable on the entire abscissa axis, it cannot be used in networks trained by the back error propagation algorithm and other algorithms that require the differentiability of the transfer function.

- *Sigmoidal transfer function* (logistic function, hyperbolic tangent, etc.) is one of the most frequently used types of transfer functions at the moment. The introduction of sigmoid-type functions was due to the limitations of neural networks with a threshold neuron activation function – with this activation function, any of the network outputs is either zero or one, which limits the use of networks not in classification tasks. The use of sigmoidal functions made it possible to switch from binary outputs of a neuron to analog ones. Transmission functions of this type, as a rule, are inherent in neurons located in the inner layers of the neural network.

  – The *logistic function* can be mathematically expressed as:

$$\sigma(x) = \frac{1}{1 + \exp(-tx)} \quad \text{where } t - \text{function parameter} \tag{2.10}$$

  When t tends to infinity, the function degenerates into a threshold one.

When $t = 0$, the sigmoid degenerates into a constant function with a value of 0.5. The range of values of this function is in the interval (0; 1). An important advantage of this function is the simplicity of its derivative:

$$\frac{d\sigma(x)}{dx} = tf(x)(1 - f(x)) \tag{2.11}$$

The fact that the derivative of this function can be expressed in terms of its value facilitates the use of this function when training a network using the backpropagation algorithm. A feature of neurons with such a transfer characteristic is that they amplify strong signals substantially less than weak ones, since the areas of strong signals correspond to gentle characteristics. This prevents saturation from large signals.

- The use of the *hyperbolic tangent function* differs from the above logistic curve in that its range of values lies in the interval (–1; 1).

$$th(Ax) = \frac{\exp(Ax) - \exp(-Ax)}{\exp(Ax) + \exp(-Ax)}, \quad \text{where} \quad th\left(\frac{A}{2}x\right) = 2\sigma(x) - 1 \tag{2.12}$$

Both graphs differ only in the scale of the axes. The derivative of the hyperbolic tangent, of course, is also expressed by a quadratic function of the value; the property of resisting saturation holds exactly the same.

- *Radial basis transfer function* takes as an argument the distance between the input vector and some pre-specified center of the activation function. The value of this function is the higher, the closer the input vector is to the center.

  In real networks, the activation function of neurons may reflect the probability distribution of a random variable, or it may indicate some heuristic dependencies between the values.

The functions listed above are only part of the set of transfer functions used at the moment. Other transfer functions include: exponent, trigonometric, modular, square, etc.

The ANN model contains several parallel structures. They are used to improve implementation efficiency on parallel architectures. There are several types of parallelization levels in artificial neural networks.

We distinguish the main levels where parallelization is carried out during the training phase:

- level of learning phase
- learning sample level
- layer level
- neuron level
- level of scales

All these levels of parallelization are in the neural network and can be used simultaneously. In order to choose the required level it is necessary to build on the number of neurons in the network, processors, features of the computer architecture of an artificial neural network and a specific task set for it.

2.4 Loss Functions

The loss function is a function that is minimized in the process of fitting the model. It represents a selected measure of disagreement between observed data and data "predicted"by the fitted function. For example, in most traditional methods for constructing general linear models, the loss function (often called least squares) is calculated as the sum of the squares of deviations from the fitted line or plane.

One of the properties (which is usually viewed as a disadvantage) of this common loss function is high sensitivity to the presence of outliers. A common alternative to minimizing the least squares loss function is to maximize the likelihood function or the logarithm of the likelihood function (or to minimize the likelihood function with a minus sign) [2].

2.4.1 Method of Least Squares

Let a regression model be given – a parametric family of functions $g(x, \alpha)$, where $\alpha \in \mathbb{R}^p$ is the vector of model parameters. We define the quality functional of the approximation of the target dependence on the sample X^l as the sum of the

squares of errors:

$$Q\left(\alpha, X^l\right) = \sum_{i=1}^{l} \left(g\left(x_i, \alpha\right) - y_i\right)^2 \tag{2.13}$$

Learning by the *method of least squares* is to find the vector of parameters α', at which the minimum of the mean square of the error is reached on a given training sample X^l:

$$\alpha' = \text{argmin} Q\left(\alpha, X^l\right) \tag{2.14}$$

The standard way to solve this optimization problem is to use the necessary minimum condition. If the function $g(x, \alpha)$ is sufficiently differentiable with respect to α, then at the minimum point a system of p equations is satisfied with respect to p unknowns:

$$\frac{\partial Q}{\partial \alpha}\left(\alpha, X^l\right) = 2\sum_{i=1}^{l}\left(g\left(x_i, \alpha\right) - y_i\right)\frac{\partial g}{\partial \alpha}\left(x_i, \alpha\right) = 0 \tag{2.15}$$

2.4.2 Non-Square Loss Function

The loss function $L(\alpha, y)$ characterizes the magnitude of the loss from the answer $\alpha \in Y$ with the exact answer $y \in Y$. It is set a priori, and thanks to it, the task of learning the algorithm $\alpha(x)$ in the sample $X^l = (x_i, y_i)^l$ reduces to minimizing the total losses:

$$Q\left(\alpha, X^l\right) = \sum_{i=1}^{l} L\left(\alpha\left(x_i\right), y\left(x_i\right)\right) \tag{2.16}$$

If the loss function is quadratic $L(\alpha, y) = (\alpha - y)^2$, then the minimization of Q corresponds to the least squares method that was considered above. For non-quadratic loss functions, numerical optimization methods are used. We will not dwell on methods in detail, but limit ourselves to listing the situations in which loss functions arise that are different from quadratic ones.

The choice of loss function is influenced by the features of the problem being solved. There is no general rule for choosing a loss function. The following loss functions are most commonly used:

- simple
- rectangular

- quadratic
- exponential etc.

Statistical estimates can be non-random (non-randomized) and random (randomize Non-randomized estimates are possible only if there is a deterministic relationship between the data obtained (the implementation) and the decision made, that is, non-random. However, the observed data is usually random. At the same time, on the basis of the accepted implementation, the probability of a particular decision is given. The choice of decision may also be random, but often this randomization is avoided.

Due to the randomness of the observed data, the decision taken (estimate) γ may not coincide with the true value of the estimated parameter l, which in the general case may be vectorial. Obviously, errors depend on the decision rule chosen. The quality of the received estimates is characterized by the loss function $C(\gamma, l)$, which is chosen so that $C(\gamma, l) \geq 0$, where the correct decision corresponds to zero values.

2.5 Backpropagation

The error backpropagation algorithm determines the weighting strategy for a multilayer network using gradient optimization methods. Currently, it is considered one of the most effective multilayer network learning algorithms.

Consider a multilayer network in which each neuron of the previous layer is connected to all the neurons of the next layer. Such a network is called fully connected. For greater generality, we set $X = \mathbb{R}^n$ and $Y = \mathbb{R}^m$ (2.6).

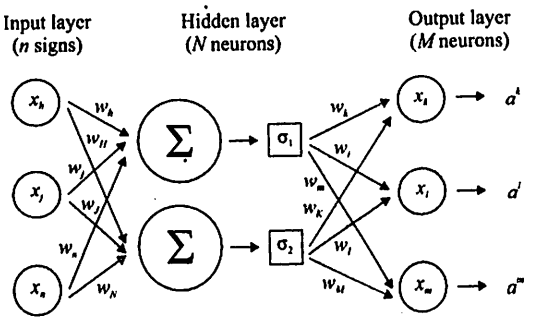


Рис. 2.6: Backpropagation

Let the output layer consist of M neurons with activation functions σ_m and outputs $a^m (m = 1, 2, \ldots, M)$. In front of it is a hidden layer of N neurons with

activation functions σ_n and outputs $a^n (n = 1, 2, \ldots, N)$. The weights of synaptic connections between the $h-th$ neuron of the hidden layer and the $m-th$ neuron of the output layer will be denoted by w_{hm}. Before this layer there can be either a distribution layer or another hidden layer with exits $v^j (j = 1, 2, \ldots, J)$ and synaptic weights w_{jh}. In general, the number of layers can be arbitrary. If there is a two-layer one, then v^j is just the $j-th$ feature: $v^j(x) = f_i(x) = x^j$ and $J = n$. Denote by w the vector of all synaptic weights of the network.

The network output values for object xi are calculated as superposition:

$$a^m (x_i) = \sigma_m \left(\sum_{n=0}^{N} w_{hm} u^n (x_i) \right) \quad \text{and} \quad u^n (x_i) = \sigma_n \left(\sum_{j=0}^{J} w_{jn} u^j (x_i) \right). \quad (2.17)$$

It is based on the objective function, formulated, as a rule, as a quadratic sum of the differences between the actual and expected values of the output signals. In the case of a single shoe sample, the target $(\bar{x}, \bar{d})$ function is defined as:

$$E(w) = \frac{1}{2} \sum_{k=1}^{M} (y_k - d_k)^2 \quad (2.18)$$

With more training samples $(j = 1, 2, \ldots, p)$, the objective function turns into a sum over all samples:

$$E(w) = \frac{1}{2} \sum_{j=1}^{p} \sum_{k=1}^{M} \left(y_k^j - d_k^j \right)^2 \quad (2.19)$$

Weights can be refined after the presentation of each training sample, or once after the presentation of all the samples that make up the training cycle.

In the future, we need partial derivatives of Q over the outputs of neurons. Write them first for the output layer:

$$\frac{\partial Q(w)}{\partial a^m} = a^m (x_i) - y_i^m = \varepsilon_i^m \quad (2.20)$$

It turns out that the partial derivative of Q with respect to a^m equals the magnitude of the error ε_i^m on the object x_i. Now write down the partial derivatives

on the outputs of the hidden layer:

$$\frac{\partial Q(w)}{\partial u^n} = \sum_{m=1}^{M} \left(a^m(x_i) - y_i^m\right) \sigma'_m w_{hm} = \sum_{m=1}^{M} \varepsilon_i^m \sigma'_m w_{hm} = \varepsilon_i^n \tag{2.21}$$

By analogy with ε_i^m, this quantity will be called the network error on the hidden layer and denoted by ε_i^n. The $\sigma_m{}'$ denotes the derivative of the activation function, calculated for the same value of the argument. If a sigmoid activation function is used, then to effectively calculate the derivative, you can use the formula:

$$\sigma'_m = \sigma_m(1 - \sigma_m) = a^m(1 - a^m(x_i)) \tag{2.22}$$

Note that ε_i^n is calculated by ε_i^m, if you start the network "back to front", passing the values $\varepsilon_i^m \sigma_m{}'$ to the outputs of the hidden layer neurons, and getting the result ε^h at the input. In this case, the input vector is scalar multiplied by the vector of weights w_{hm}, located to the right of the neuron, and not to the left, as in the direct calculation (hence the name of the algorithm – the reverse propagation of errors).

2.6 Optimization Algorithms

When training the neural network, various optimization algorithms are used, most of them are an improvement of the stochastic gradient descent.

Training of the neural network is the selection of parameters (weights) in such a way as to minimize the penalty function on the training set. Formalizing, we have:

$$\begin{aligned} X^{(k)} &= \{x_i | i = Mk, ..., (Mk + m - 1)\} \\ Y^{(k)} &= \{y_i | i = Mk, ..., (Mk + m - 1)\} \end{aligned} \tag{2.23}$$

So,

$$L^{(k)}(w) = \sum_{i=0}^{M-1} L(w, xMk + 1, yMk + 1), \quad \text{where } k = 0, 1, \ldots, (K\text{-}1) \tag{2.24}$$

And instead of one iteration of the gradient descent, we get a set of K mini-

iterations of the form:

$$w_p^{(k+1)} = w_p^{(k)} - \eta \cdot \nabla L^{(k)}\left(w_p^{(k)}\right), \quad \text{where} \quad w_{p+1}^{(0)} = w_p^{(K)} \tag{2.25}$$

Each "big" iteration ($p = 0, 1, 2, \ldots$), when we go through the whole set, will be called an *epoch*. Between the ages, the training set is mixed so that the elements fall into different mini batches in different eras. So $L^{(k)}(w)$ are generally summarized over different subsets of X, Y for different epochs (for fixed k).

To solve this problem, an approach was proposed that increases the step towards the minimum and reduces oscillation. This is achieved due to the fact that the parameter shift is calculated as the weighted sum of the shift in the previous step and the new one based on the gradient:

$$\begin{aligned} \Delta_{p+1} &= \gamma \cdot \Delta_p + \eta \cdot \nabla L\left(w_p\right) \\ w_{p+1} &= w_p - \Delta_{p+1} \end{aligned} \tag{2.26}$$

The speed of movement in the direction of the minimum increases (since this direction is present in all gradients), and the oscillation is extinguished. The weight parameter γ is usually chosen to be 0.9 or close to that.

2.6.1 Nesterov Accelerated Gradient (NAG)

This is the next improvement of the stochastic gradient descent method, increasing the rate of convergence. Instead of calculating the gradient at the current point, we will use the gradient at the "predicted"point based on the shift calculated in the previous step.

The case of "pure"moments for illustration can be seen in Figures 2.7 and 2.8.

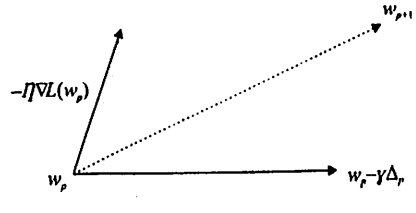


Рис. 2.7: 1

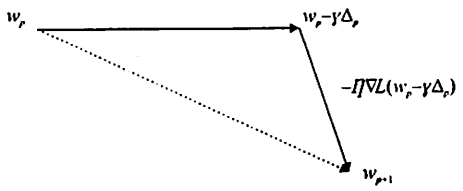


Рис. 2.8: 2

Formally, this change will be written as:

$$\begin{aligned} \Delta_{p+1} &= \gamma \cdot \Delta_p + \eta \cdot \nabla L\left(w_p - \gamma \Delta_p\right) \\ w_{p+1} &= w_p - \Delta_{p+1} \end{aligned} \tag{2.27}$$

Here we proceed from the assumption that the first term makes the main contribution to the shift vector, and the term with the gradient only "clarifies". It is therefore logical to calculate the refinement gradient in the vicinity of the new point, and not in the current one.

2.6.2 Adaptive Gradient Algorithm (AdaGrad)

So far we have used the uniform learning rate η. However, when we train a neural network with millions of parameters, confinence will be decreased. Some parameters will change more often, some less. Intuitively, it seems that parameters that rarely change need to be changed with greater speed than those that change often. We use this algorithm, which adjusts the learning rate of each of the parameters, depending on how much these parameters have changed before.

Instead of a single scalar η as the learning rate, at each iteration we will define the vector $\eta_p = (\eta_p^{(1)}, \eta_p^{(2)}, \ldots, \eta_p^{(d)})$. In this case, the η that we used earlier is just the initial learning rate. For the first iteration, we set $\eta_p^{(i)} = \eta (i = 1, 2, \ldots, d)$.

So, for each iteration p, we have the current parameter vector:

$$G_p = \left(G_p^{(1)}, G_p^{(2)}, ..., G_p^{(d)}\right) \tag{2.28}$$

Each element is the sum of squares of partial derivatives of the error function with respect to the corresponding parameter, i.e.

$$G_p^{(i)} = \sum_{j=1}^{p} \left(g_j^{(i)}\right)^2, \quad \text{where } i = 1, 2, \ldots, d \tag{2.29}$$

Finally, the components of the learning velocity vector η_p are defined as:

$$\eta_p^{(i)} = \frac{\eta}{\sqrt{G_p^{(i)} + \varepsilon}} \tag{2.30}$$

ε– the parameter that insures dividing by zero, usually about $1e-8$ is chosen. We change the parameters using almost the same formula as before:

$$w_{p+1} = w_p - \eta_p \bigodot \nabla L\left(w_p\right) \tag{2.31}$$

Only instead of multiplying the scalar by the vector, we multiply the two vectors element wise (denoting this operation $\bigodot$).

2.6.3 Adaptive Learning Rate Method (Adadelta)

The main problem of *AdaGrad* is that the denominator in the coefficient of learning speed is constantly increasing, respectively, after some time for some of the parameters the learning rate will drop to zero. It is also extremely important to choose "good"initial values of the parameters, because if at the beginning of training we get large gradients for some parameters, then the learning rate will decrease and will not increase in the future, despite the change in the parameters and their gradients. *AdaGrad* also does not save us from having to choose the initial value of η.

The idea of a method of dealing with a continuous decrease in the speed of learning is as follows. Replace the sum of the squares of all partial derivatives with respect to a certain parameter, with the sum of only the last W derivatives. More precisely, replace the l^2 norm:

$$\sqrt{\sum_{j=1}^{p}\left(g_j^{(i)}\right)^2}, \quad \text{where } i = 1, 2, \ldots, d \tag{2.32}$$

on the root mean square of the last W derivatives:

$$RMS_p^{(i)} = \sqrt{\frac{1}{W}\sum_{j=p-W}^{p}\left(g_j^{(i)}\right)^2}, \quad \text{where } i = 1, 2, \ldots, d \tag{2.33}$$

It seems in this case we get rid of the problem of both a constant decrease in the speed of learning, and the fact that, if the initial values for some parameters are unsuccessful, the partial derivatives will be too large. However, to apply the idea directly in this form is inconvenient purely technically. We will need to store the W of the last gradients, and W should be large enough for it to make sense. Therefore, the arithmetic average of squares of partial derivatives is proposed to be approximated using an exponentially weighted moving average:

$$EG_{p+1}^{(i)} = \gamma \cdot EG_p^{(i)} + (1-\gamma) \cdot \left(g_p^{(i)}\right)^2 \tag{2.34}$$

for a certain selection of γ, we can assume:

$$RMS_p^{(i)} \approx \sqrt{EG_p^{(i)}} \tag{2.35}$$

Thus, replacing the formula for changing parameters from *AdaGrad* by:

$$w_{p+1} = w_p - \frac{\eta}{\sqrt{EG_{p+1} + \varepsilon}} \bigodot \nabla L\left(w_p\right) \tag{2.36}$$

we get rid of the constant decrease in the speed of learning.

There remains a problem with the choice of the parameter η – the initial learning rate. Denote the change of the $i-th$ parameter at the p step as $\Delta_p^{(i)}$ and, accordingly, the change of parameters as:

$$w_p = w_{p-1} - \Delta_p \tag{2.37}$$

in addition to calculating the moving average of the squares of the gradients, we will also attend to the calculation of the moving average :

$$E\Delta_p^{(i)} = \gamma \cdot E\Delta_{p-1}^{(i)} + (1-\gamma) \cdot \left(\Delta_p^{(i)}\right)^2 \tag{2.38}$$

and we will change the parameters as:

$$\Delta_p = \sqrt{\frac{E\Delta_{p-1} + \varepsilon}{EG_p + \varepsilon}} \bigodot \nabla L\left(w_p\right) \tag{2.39}$$

2.6.4 Adaptive Moment Estimation (Adam)

As in *Adadelta*, these moving averages approximate the first and second moment of the gradients. v_p is proposed to be used to change the learning rate, and m_p to replace the usual gradient calculated on the current set of parameters. The latter is based on the idea, namely to preserve the inertia of the shift of parameters from previous iterations.

Since the initial values m_0, v_0 are initialized with zeros, and the coefficients β_1, β_2 are chosen close to unity, the "moments" preserve the shift to zero. To overcome this problem, it is proposed to change the parameters instead of directly m_p, v_p use:

$$\widehat{m_p} = \frac{m_p}{1-\beta_1^p} \quad \text{and} \quad \widehat{v_p} = \frac{v_p}{1-\beta_2^p} \tag{2.40}$$

The formula for changing parameters is written in the form:

$$w_p = w_{p-1} - \frac{\eta \cdot \widehat{m_p}}{\sqrt{\widehat{v_p}} + \varepsilon} \tag{2.41}$$

2.7 Recurrent Neural Network

One of the popular models of artificial neural networks is the recurrent neural network (RNN). Unlike the traditional neural network, it contains feedback and allows you to save information using its internal memory. Due to this, the RNN can process a series of events in time or consecutive spatial chains. RNN are also widely used in the field of natural language processing, speech generation and machine translation. The structure of the RNN is shown in 2.9.

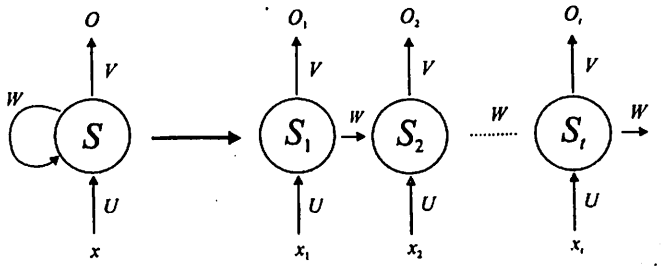


Рис. 2.9: Recurrent Neural Network

Here x_i is the input; O_i – output data; S_i – the hidden state that performs the function of storage; U, V, W – parameters of the neural network.

The diagram shows a deployed RNN. This is done to show that RNN are used to process time sequences. So, for example, if there is a problem of processing sentences of natural language, then the number of layers of the neural network will correspond to the number of words in the sentence.

One of the advantages of using RNN is that they can be used to associate previous information with the current task. For example, when analyzing a current video frame, information about previous frames may be useful. Indeed, if the current task requires recent information, the use of a RNN is justified and will lead to the desired result.

Formally, training of the neural network has the form (x, y):

$$x = \{x_i | i = 0, 1.....,T\}, \quad \text{where } x(t) \text{ - feature vector} \tag{2.42}$$

$$y = \{y_i | i = 0, 1.....,T\}, \quad y(t) = \{y_1(t), y_2(t), ..., y_N(t)\} \tag{2.43}$$

Here $y_k(t) = 1$ and $\forall j \neq k\,(y_j(t) = 0)$, k - class number, t - sequence index.

To use a recurrent neural network in solving certain problems, the main phases that should be paid priority attention are training and direct work with data. Learning as a phase consists of setting up the network parameters (for example, weight and activation threshold in the activation function), while acting based on the learning algorithm.

There are three main classes of learning algorithms:

- learning with a teacher;
- learning without a teacher;
- learning algorithm with fixed weights.

In the first case, when training with a teacher, the algorithm includes the data of the incoming vector, as well as the set of the expected output vector. From the so-called training vector in conjunction with the corresponding output vector for it, a training pair is made, where the expected output vector implements the correction function. Thus, the number of its functions includes the correction of weights, to the state of compliance of the current output with the expected one. It is necessary to bring their values as close as possible. The great advantage of

neural networks with a teacher is their iteration and the ability to produce the result of the most acceptable level. The system will not stop work until the results provided by the network result in an acceptable level of discrepancy. One of the most popular learning algorithms in artificial neural networks with a teacher is the use of the back-propagation error method.

In the second class of learning algorithms, which is an artificial neural network with a learning algorithm without a teacher, only input vectors are included in the training set. This type of algorithm is iterative. If the same inputs match the same outputs, the neural network stops learning, since the whole system takes a more organized form. One example of artificial neural networks using such a teaching algorithm without a teacher is self-organizing maps.

In the third case of learning algorithms, when a learning algorithm with fixed weights is used, neural networks determine weights by performing some predefined calculations using training vectors. In the case of using Hopfield neural networks (using bidirectional associative memory), the predefined learning algorithms operate much faster than the first two classes of learning algorithms. It is necessary to emphasize that, according to the results provided, unsupervised learning algorithms are more reliable because they provide more distinct data than neural networks that use a learning algorithm with a teacher that compares the current output and the desired result.

2.7.1 Elman neural network

Elman neural network, or a simple recurrent perceptron. This is one of the simplest configurations of a RNN, consisting layers.

- Input layer;
- Simple recurrent layer;
- Output layer.

Such a network is given by the following relations:

$$net^{(i)}(t) = Vx(t) + Us(t-1), \quad s(t) = f\left(net^{(i)}(t)\right) \tag{2.44}$$

$$net^{(j)}(t) = Ws(t), \quad o(t) = g\left(net^{(j)}(t)\right) \tag{2.45}$$

We also denote by $h_w(x,t) = o(t)$ activation of the output layer at step t of the RNN under the condition that x - input sequence at a specific value $w = (U, V, W)$. Also denote $h_w(x) = \{h_w(i,x) | i = 1, 2, \ldots, T\}$.

2.7.2 Hopfield neural network

It consists of a single layer of neurons, the number of which is simultaneously the number of inputs and outputs of the network. Each neuron is connected by synapses with all other neurons, and also has one input synapse, through which the signal is input. Output signals, as usual, are formed on axons. The block diagram of the Hopfield neural network is shown in 2.10.

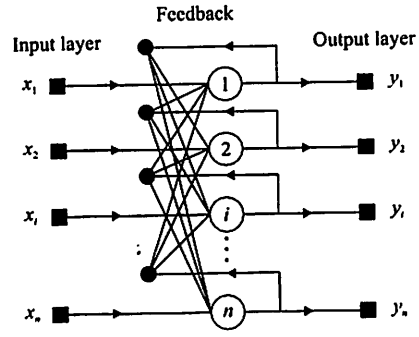


Рис. 2.10: Hopfield Neural Network

The problem solved by this network as an associative memory, as a rule, is formulated as follows. A certain set of binary signals (images, sound digitization, other data describing certain objects or characteristics of processes), which are considered exemplary, is known. The network should be able to select ("remember"by partial information) the corresponding sample (if any) or "give a conclusion"that the input data does not correspond to any of the samples from an arbitrary non-ideal signal applied to its input. In general, any signal can be described by the vector $X = \{x_i | i = 0, 1, \ldots, n-1\}$, n is the number of neurons in the network and the dimension of the input and output vectors. Each element x_i is either $+1$ or -1. Denote the vector describing the $k-th$ pattern by X^k, and its components, respectively x_i^k $(k = 0, 1, \ldots, m-1)$, m is the number of samples. When the network recognizes (or "remembers") any sample based on the data presented to it, its outputs will contain it, that is $Y = X^k$, where Y is the vector of output

network values: $Y = \{y_i | i = 0, 1, \ldots, n-1\}$. Otherwise, the output vector will not match any exemplary.

At the network initialization stage, the weights of synapses are set as follows:

$$w_{ij} = \begin{cases} \sum_{k=0}^{m-1} x_i^k x_j^k, & (i \neq j) \\ 0, & (i = j) \end{cases} \tag{2.46}$$

Here, i and j are the indices, respectively, of the presynaptic and postsynaptic neurons; x_i^k and x_j^k - $i-th$ and $j-th$ elements of the vector of the $k-th$ sample.

As mentioned above, sometimes the network can not conduct recognition and outputs a non-existent image. This is due to the problem of limited network capabilities. For the Hopfield neural network, the number of memorized images m should not exceed a value approximately equal to $0.15 * n$. In addition, if the two images A and B are very similar, they will probably cause cross associations in the network, that is, presentation of the vector A to the network inputs will result in the appearance of vector B at its outputs and vice versa.

2.7.3 Hamming neural network

When there is no need for the network to explicitly produce a sample, that is, it is enough, say, to get a sample number, the associative memory is successfully implemented by the Hamming network. This network is characterized, compared to the Hopfield neural network, by lower memory costs and computational volume, which is evident from its structure (2.11).

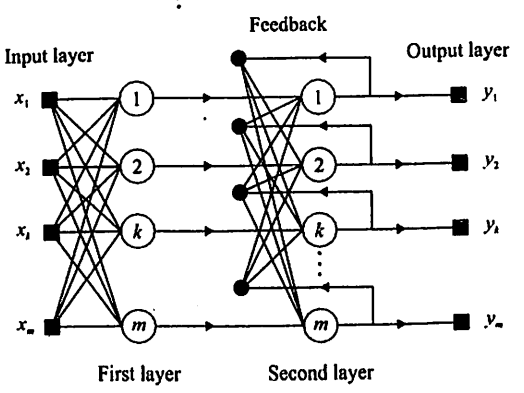


Рис. 2.11: Hamming Neural Network

The network consists of two layers. The first and second layers each have m neurons, where m is the number of samples. The neurons of the first layer have n

synapses each connected to the network inputs (forming a dummy zero layer). The neurons of the second layer are interconnected by inhibitory (negative feedback) synaptic connections. A single synapse with positive feedback for each neuron is connected to its own axon.

The idea of the network is to find the Hamming distance from the test image to all samples. The Hamming distance is the number of distinct bits in two binary vectors. The network must select a sample with a minimum Hamming distance to an unknown input signal, with the result that only one network output will be activated corresponding to this sample [3].

At the initialization stage, the weights of the first layer and the threshold of the activation function are assigned the following values:

$$w_{ik} = \frac{x_i^k}{2} \quad \text{and} \quad T_k = \frac{n}{2} \tag{2.47}$$

where $i = \{0, 1, \ldots, n-1\}$ and $k = \{0, 1, \ldots, m-1\}$, hence x_i^k - $i{-}th$ element of the $k-th$ sample.

The weighting factors of the inhibitory synapses in the second layer are taken equal to a certain value $0 < e < m^{-1}$. The synapse of a neuron associated with its axon has a weight of $+1$.

2.8 Long Short-Term Memory

Long short-term memory (LSTM) is a kind of RNN capable of learning long-term dependencies. Its difference from the standard RNN is in the structure of the repeating module of the neural network. Instead of a single layer used in a RNN, LSTM uses four layers that interact with each other in a special way. The structure of the repeating module of the LSTM network is shown in 2.12.

On the diagram, the rectangle shows the layers of the neural network, the oval - point wise operations. The main element of the LSTM network is the cell state, which is a horizontal line at the very top of the circuit. The state of the cell passes directly through the whole chain, participating in some linear transformations. LSTM can remove information from the state of a cell using filters that are implemented as a sigmoidal neural network layer and a point wise multiplication operation. In total, there are three such filters in the LSTM network

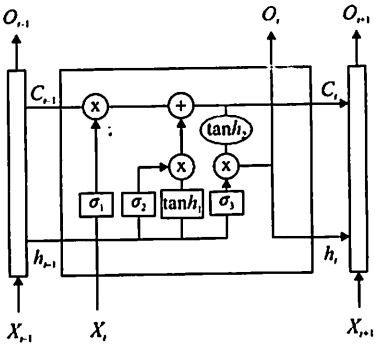


Рис. 2.12: Long Short-Term Memory

module.

The operation of the LSTM-network module can be described by the following algorithm:

- Definition of information that can be removed from the state of the cell. This action performs the σ_1 layer, which processes the input data h_{t-1} and x_t and returns a coefficient from 0 to 1. The smaller this coefficient, the more information will be deleted.

- Determining the information to be updated in the cell state. The layer σ_2 is responsible for this action.

- Construction of a vector of new values that are added to the state of the cell. It is performed by h_1 layer.

- Formation of the output information. First, the layer σ_3 is applied, which decides which information from the state of the cell will be output. Then the state values of the cell pass through the h_2 layer in order to obtain values from the range from -1 to 1 at the output, and are multiplied with the output values of the sigmoidal layer, which allows only the required information to be output.

LSTM networks are a significant step in the development of RNN. Their use avoids the problem of long-term dependencies by expanding the range of tasks solved using machine learning methods. LSTM networks have taken their place among other ANN, becoming the most popular solution in the field of natural language processing.

At each step of learning t, the value $h^{(t)} \in \mathbb{R}^m$ of the hidden layer of the RNN is calculated as follows:

$$h^{(t)} = f\left(Wx^{(t)} + Uh^{(t-1)} + b^h\right) \tag{2.48}$$

where $x^{(t)} \in \mathbb{R}^m$ – input vector at time t (for example, the vector representation of the current word in the text fragment),$W \in \mathbb{R}^{m\times n}$,$U \in \mathbb{R}^{m\times n}$,$b^h \in \mathbb{R}^m$ – learning parameters of the recurrent neural network, f – nonlinear transform function.

Most often, one of the following functions is used as a nonlinear transformation:

- Sigmoid function: $f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic tangent: function: $f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Rectifier function: $f(x) = \max(0, x)$

In RNN, the output value $y^{(t)} \in \mathbb{R}^l$ at the current step t is calculated by the formula:

$$y^{(t)} = W^{\tilde{n}} h^{(t)} + b^{\tilde{n}} \tag{2.49}$$

where $W^c \in \mathbb{R}^{l\times m}$ and $b^c \in \mathbb{R}^l$ – learning parameters.

The data used for training should be:

- Representative, that is, comprehensively illuminate the object being modeled, while redundancy is acceptable.
- Qualitatively mixed by an algorithm that uses a normal distribution of random variablesn a simple.

2.9 Self-Generating Neural Network

The basis for the self-generating neural networks (SGNN) is the observed pattern that global network ordering becomes possible as a result of self-organizing operations that are independently carried out in different local network segments. In accordance with the submitted signals, neurons are activated, as a result of which one neuron in the network, which is called the *output neuron*, is active.

Among the mechanisms of SGNN can be divided into two main classes:

- Self-organization based on *Hebbian associative rule*;
- The mechanism of competition of neurons based on the *Kohonen learning rule.*

Neurons are placed in lattice sites, usually one- or two-dimensional. Higher dimension networks are also possible, but are used quite rarely. As a rule, these are single-layer networks of direct propagation, in which a neuron is connected to all components of N-dimensional input vector x as it is schematically depicted for $N = 2$ in 2.13.

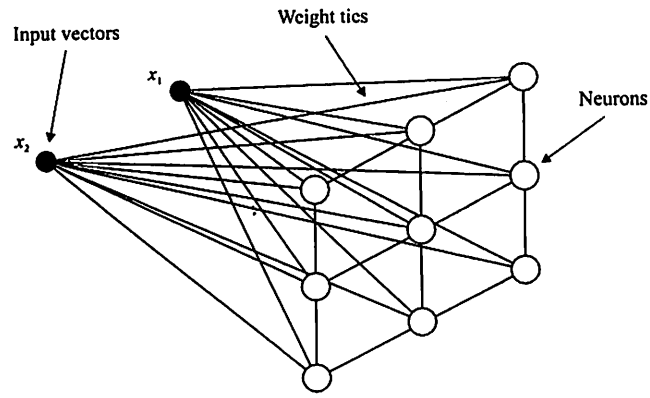


Рис. 2.13: Self-Generating Neural Network

The formation of SGNN begins with the initialization of the network's synaptic weights. Usually, synaptic weights are assigned small values that are generated by a random number generator. With such initialization, the network initially does not have any order of attributes of the input vectors.

The weights of the synaptic connections of neurons form a vector $w_i = \{w_{i1}, w_{i2}, \ldots$. After normalization of input vectors upon activation networks by the vector x in competition wins the neuron whose weights differ least of all from the corresponding components of this vector. For the $w - th$ neuron-winner, the relation

$$\{d(x, w_w) = \mathrm{mind}(x, w_i) \,|i = 1, 2, ..., K\} \tag{2.50}$$

where $d(x, w)$ denotes the distance between the vectors x and w, and K is the number neurons. Topological around the winning neuron the neighborhood $S_w(t)$ with a certain energy decreasing over of time. The winning neuron and all neurons lying within its vicinity undergo adaptation, during which their weights

vectors change in the direction of the x vector according to *Kohonen rule*:

$$w_i(t+1) = w_i(t) + \eta_i(t)(x - w_i(t)) \quad \text{for} \quad i \in S_w(t) \tag{2.51}$$

where $\eta_i(t)$ – learning coefficient of the i-th neuron in the neighborhood $S_w(t)$ at the $t-th$ moment of time. The value of $\eta_i(t)$ decreases with increasing the distance between the $i-th$ neuron and the winner. Weights of neurons located outside the neighborhood $S_w(t)$, do not change. Neighborhood size and odds neuron learning are functions whose values decrease over time. Adaptation by formula is equivalent to the gradient learning method based on minimizing the objective function:

$$E(w) = \frac{1}{2}\sum_{i,j,t} S_i(x(t))[x_j(t) - w_{ij}(t)]^2 \tag{2.52}$$

and $S_i(x(t))$ is a neighborhood definition function, changing in the learning process.

After presenting two different vectors x_1 and x_2 are activated two neurons of the network whose weights are closest to the coordinates corresponding vectors. These weights, denoted w_1 and w_2 can display in space as two points. Convergence of vectors x_1 and x_2 causes a corresponding change in the location of the vectors w_1 and w_2 in limit equality $w_1 = w_2$ is performed then and only when x_1 and x_2 coincide or almost indistinguishable from each other. The network in which these conditions are met is called a topographic map or a *Kohonen map* [4].

2.9.1 Algorithm WTA

The learning algorithms used to train *Kohonen neural networks* are called *unsupervised learning algorithms*. Similar algorithms are used when there are no reference output values for the input vectors.

The purpose of training a network with self-organization based on competition is considered to be such an ordering of neurons that minimizes the value of the deviation of the weight vector from the input vector x. For p input vectors x, this error in the Euclidean metric can be expressed as:

$$Eq = \frac{1}{2}\sum_{t=1}^{p} ||x_i - w_{w(t)}||^2 \tag{2.53}$$

where $w_{w(t)}$ - weight of the winning neuron of the vector x.

This approach is also called *vector quantization*. The numbers of the winning neurons in the successive presentation of vectors x constitute the so-called code table. In the classical solution of the coding problem, the K-averaging algorithm, named after the generalized *Lloyd algorithm*, is applied.

For neural networks, the analogue of the *Lloyd algorithm* is considered to be the WTA algorithm (Winner Takes All). In accordance with it, after the presentation of the vector x, the activity of each neuron is calculated. The winner is the neuron with the strongest output signal, then is the one for which the scalar product $x^T w$ is the greatest that for normalized vectors is equivalent to the smallest Euclidean distance between the input vector and the vector of weights of neurons. The winner gets the right to clarify their weight in the direction of the vector x according to the rule:

$$w_w(t+1) = w_w(t) + \eta(x - w_w(t)) \tag{2.54}$$

Weights of the remaining neurons are not subject to specification. The algorithm allows to take into account the fatigue of neurons by counting the number of victories of each of them and to encourage the elements with the least activity to equalize their chances.

In addition to WTA algorithms, in which only one neuron can be trained in each iteration, algorithms such as WTM (Winner Takes Most) are used to train networks with self-organization, in which, in addition to the winner, they specify the values of their weights and neurons from its nearest environment. At the same time, the further a neuron is located from the winner, the less its weight changes.

The process of refining the weight vector can be defined as:

$$w_i(t+1) = w_i(t) + \eta_i S(i,x)(x - w_i(t)) \tag{2.55}$$

for all i neurons located in the victor's neighborhood. At given formula, the learning coefficient η_i of each neuron is separated from its distance to the presented vector x is determined by the function $S(i,x)$. If $S(i,x)$ defined in the form:

$$S(i,x) = \begin{cases} 1, & for\ i = w \\ 0, & for\ i \neq w \end{cases} \tag{2.56}$$

for w denotes the winner number, then we get the classic algorithm WTA. There are many variants of the WTM algorithm that differ, first of all by the form of the function $S(i,x)$.

2.9.2 Kohonen algorithm

Kohonen algorithm belongs to the oldest algorithms for training networks with self-organization based on competition, and currently there are various versions of it. In the classical *Kohonen algorithm*, the network is initialized by assigning certain positions in space to neurons and associating them with neighbors on an ongoing basis. At the moment of choosing the winner, not only his weights are specified, but also the weights and his neighbors in the immediate vicinity. In this way, the winning neuron is being adapted along with its neighbors.

$$S(i,x) = \begin{cases} 1, & for \quad -K < d(i,w) < K \\ 0, & else \end{cases} \tag{2.57}$$

In this expression $d(i,w)$ may denote as Euclidean distance between the vectors of the weights of the winning neuron w and the $i-th$ neuron, and the distance, measured by the number of neurons.

Another type of neighborhood in *Kohonen maps* is the Gaussian neighborhood, the type at which the function $S(i,x)$ determined by the formula:

$$S(i,x) = \exp\left(-\frac{d^2(i,w)}{2\lambda^2}\right) \tag{2.58}$$

The refinement of the weights of neurons occurs according to the rule:

$$w_i(t) = w_i(t-1) + \eta(t) \cdot S(i,w)(x - w_i) \tag{2.59}$$

The degree of adaptation of neighboring neurons is determined not only by the Euclidean distance between the $i-th$ neuron and the winning neuron ($w-th$ neuron) $d(i,w)$, but also by the level of neighborhood λ. Generally Gaussian the neighborhood gives better learning outcomes and provides better networking than a rectangular neighborhood.

3. Sentiment Analysis

Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to identify and extract subjective information in source materials.

Sentiment analysis is becoming an increasingly important task, both from theoretical and applied points of view.

Our goal is to test and compare, firstly, various approaches to the presentation of text within a vector space model, secondly, several machine learning methods, including the *Support Vector Machine*, the *Naive Bayes* classifier, keyword based classification method and its combination with SVM.

3.1 Vector Space Model

The main idea of the *Vector Space Model* (VSM) is the representation of each document of the collection as a point in multidimensional space. The points close to each other correspond to semantically similar documents. In search engines, this idea is implemented as follows. User request is considered as a vector (as a pseudo-document). Then, the documents are sorted in ascending order of the distance to the pseudo-document and are given to the user.

The VSM is based on the hypothesis of statistical semantics: the statistical dependencies of the use of words by man can be used to find the meaning embedded in them.

The text is considered as a structure, that is, as a collection of separate interrelated sentences, combined into a subset of paragraphs, paragraphs, chapters, etc. This structure provides an expression of the main idea, the purpose of writing this text by the author, through many subgoals of different rank, defined by individual sentences, paragraphs, paragraphs, chapters, etc.

Below are the mathematical interpretations of the vector model of a complete element of the text, consisting, for example, of sections containing paragraphs, which, in turn, consist of sentences, that is:

$$\begin{aligned} G &= \{G_1, G_2, ..., G_i, ..., G_n\} \\ V_g &= \{V_{g_1}, V_{g_2}, ..., V_{g_i}, ..., V_{g_n}\} \end{aligned} \tag{3.1}$$

where G - set of sections, G_i - $i-th$ section $\{i = 1, 2, \ldots, n\}$, Vg - set of sections with vector goals, V_{g_i} - $i-th$ section with vector goal.

In turn,

$$\begin{aligned} A_i &= \{A_{i_1}, A_{i_2}, ..., A_{i_j}, ..., A_{i_m}\} \\ V_{a_i} &= \left\{V_{a_{i_1}}, V_{a_{i_2}}, ..., V_{a_{i_j}}, ..., V_{a_{i_m}}\right\} \end{aligned} \tag{3.2}$$

where A_i - set of paragraphs of $i-th$ section, A_{i_j} - $j-th$ paragraphs of $i-th$ section $\{j = 1, 2, \ldots, m\}$, V_{a_i} - set of paragraphs with vector goals, $V_{a_{i_j}}$ - $j-th$ paragraphs of $i-th$ section with vector goal.

The mathematical interpretation of the vector model for sentences is expressed as:

$$\begin{aligned} P_{i_j} &= \left\{P_{i_{j_1}}, P_{i_{j_2}}, ..., P_{i_{j_h}}, ..., P_{i_{j_k}}\right\} \\ V_{P_{i_j}} &= \left\{V_{P_{i_{j_1}}}, \mathrm{V}_{P_{i_{j_2}}}, ..., \mathrm{V}_{P_{i_{j_h}}}, ..., \mathrm{V}_{P_{i_{j_k}}}\right\} \end{aligned} \tag{3.3}$$

where P_{i_j} - set of sentences of $i-th$ section $j-th$ paragraph, $P_{i_{j_h}}$ - $h-th$ sentence of $j-th$ paragraph of $i-th$ section $\{h = 1, 2, \ldots, k\}$, $V_{p_{i_j}}$ - set of sentences of $i-th$ section $j-th$ paragraph with vector goals, $V_{p_{i_{j_h}}}$ - $h-th$ sentence of $j-th$ paragraph of $i-th$ section with vector goal.

From the presented description it is clear that for each element (fragment) of the text some vector of the goal is put in correspondence. As is well known, the semantic and grammatical center of a sentence is usually a predicate expressed by a verb (full-valued or a bunch). Under certain conditions, nominal groups (nouns with or without dependent words) can act as a complete sentence. Examples of such sentences, called nominal or nominative. For example:

Twenty first. Night. Monday. Outlines of the capital in the mist.

In addition to nominal sentences, incomplete sentences can also be considered, which are formed from complete ones by certain abbreviations. For example:

Screwdriver! (instead of "*Give me a screwdriver*!").

The reasons for such a reduction, called an ellipse, can be varied, but usually

the part of the sentence that is designed for a certain knowledge of the hearer is reduced.

As noted above, each sentence carries a certain meaning, laid by the author, and provides advancement to the final goal, as the main idea in the context of the goal system of the semantic group of sentences, paragraph, etc. In general, each sentence has a corresponding goal vector. Thus, the text can be defined as a structure of interrelated concepts, ensuring progress towards the final goal expressed by the author's idea.

The proposed target vector model can be represented as three components:

- *Vbegin* – the initial goal, expressed in terms of the initial vector $\vec{X}$ with the given coordinates;
- *Vend* – the final goal, expressed in terms of the final vector $\vec{Y}$ with given coordinates;
- $\vec{Z}$ is the type of connection between the initial vector $\vec{X}$ and the final vector $\vec{Y}$.

The coordinates of the vector can be individual words, concepts, nominal groups, separate sentences, semantic groups of sentences, paragraphs, etc.

Since there are three components that define a vector, respectively, for the subsequent analysis we have identified the following types of vectors:

- *Simple Vector*: $\vec{V} = \left(\vec{X}\right)$ or $\vec{V} = \left(\vec{Y}\right)$;
- *Zero Vector*: $\vec{V} = (\varnothing)$;
- *Full Vector*: $\vec{V} = \left(\vec{X}, \vec{y}\right)$ with a bond $\vec{X}$;
- *Empty Vector*: $\vec{V} = \left(\vec{X}, \vec{y}\right)$ without a bond $\vec{X}$;
- *Left Vector*: $\vec{V} = \left(\vec{X}\right)$;
- *Right Vector*: $\vec{V} = \left(\vec{Y}\right)$.

In turn, the vectors $\vec{X}$ and $\vec{Y}$ can consist of sub-vectors, as separate independent parts – coordinates belonging to the current proposal.

Each coordinate has its own attributes. Attributes can be temporary or spatial characteristics of the coordinates.

The composition of the vector coordinates is determined by the complexity of constructing the sentence. In the general case, the hierarchy of individual parts of a sentence can be eliminated by normalization.

Consider the application of the vector model described above with a specific example.

Let's analyze the following sentence:

At all times, people are faced with the same problems of the economy.

This sentence can be presented in a vector form: $V_p(x_1, y_1)$ with a link of the form z_1, or in a simplified form $V_p(x_1, y_1)(z_1)$, where coordinate x_1 = people, coordinate y_1 = problems of the economy, connection of the form z_1 = are faced.

The attributes of the x_1 coordinates are atr_x = at all times, and the attributes of the y_1 coordinates are atr_y = the same.

On the basis of the vector representation, some problems of processing textual information can be solved, in particular:

- reduction of the amount of initial information for the execution of text analysis procedures and the formation of systems and knowledge bases;
- text synthesis using information extracted from knowledge bases.

3.2 Support Vector Machines

Support Vector Machine (SVM) is one of the most popular teaching methods used to solve classification and regression problems. The main idea of the method is to build a hyperplane that separates the objects of the sample in the most optimal way. The algorithm works under the assumption that the greater the distance (gap) between the separating hyperplane and the objects of the divided classes, the smaller will be the average classifier error.

Consider the classification problem into two non-intersecting classes in which objects are described by n-dimensional real vectors: $X = \mathbb{R}^n, Y = \{-1, +1\}$.

We will build a linear threshold classifier:

$$a(x) = sign\left(\sum_{j=1}^{n} w_j x^j - w_0\right) = sign\left(\{w, x\} - w_0\right) \tag{3.4}$$

where $x = (x^1, x^2, \ldots, x^n)$ - feature description of the object x,
$w = (w^1, w^2, \ldots, w^n) \in \mathbb{R}^n$ and scalar threshold $w_0 \in \mathbb{R}^n$ are the parameters of the algorithm.

The equation $\{w, x\} = w_0$ describes a hyperplane separating classes in the space $\mathbb{R}^n$.

Suppose that the sample is linearly separable, that is, there are values of the parameters w, w_0 for which the number of errors functional:

$$Q(w, w_0) = \sum_{i=1}^{l} [y_i (\{w, x_i\} - w_0) < 0] \tag{3.5}$$

takes a zero value. But then the separating hyperplane is not unique, since there are other provisions of the separating hyperplane that implement the same partitioning of the sample. The idea of the method is to wisely manage this freedom of choice. We require that the separating hyperplane be as far as possible from the points of both classes nearest to it. Initially, this classification principle arose from heuristic considerations: it is quite natural to assume that maximizing the gap (margin) between classes should contribute to a more confident classification. In the future, this principle has received a powerful theoretical justification.

Normalization. Note that the parameters of the linear threshold classifier are determined up to normalization: the algorithm $a(x)$ does not change if w and w0 are simultaneously multiplied by the same positive constant. It is convenient to choose this constant in such a way that the conditions

$$\{w, x_i\} - w_0 = y_i \tag{3.6}$$

are satisfied for all the boundary (i.e. objects closest to the separating hyperplane) x_i from X^l.

This is possible because, at the optimal position of the dividing hyperplane, all boundary objects are at the same distance from it. The remaining objects are on. Thus, for all $x_i \in X^l$:

$$\{w, x_i\} - w_0 \begin{cases} \leq -1, & if\ y_i = -1 \\ \geq 1, & if\ y_i = +1 \end{cases} \tag{3.7}$$

Condition $-1 < \{w, x\} - w_0 < 1$ defines a strip separating classes. None of

the points in the training set can lie within this band. The borders of the strip are two parallel hyperplanes with the guiding vector w. The points closest to the separating hyperplane lie exactly at the boundaries of the strip. Moreover, the hyperplane itself separates exactly in the middle of the strip.

Dividing Strip Width. In order for the separating hyperplane to be as far as possible from the sampling points, the width of the strip must be maximal. Let x_- and x_+ be two arbitrary points of classes 1 and +1, respectively, lying on the boundary of the strip. Then the width of the strip is:

$$\left\{ \{x_+ - x_-\}, \frac{w}{||w||} \right\} = \frac{\{w, x_+\} - \{w, x_-\}}{||w||} = \frac{2}{||w||} \tag{3.8}$$

The width of the strip is maximum when the norm of the vector w is minimal.

3.3 Naive Bayes Classifier

The *Naive Bayes classifier* is a family of classification algorithms that take one assumption: Each parameter of the data to be classified is considered independently of the other class parameters.

Theoretically, the *Naive Bayes classifier* provides an error comparable to other types of classifiers, such as neural networks and a decision tree. However, in practice its use may be limited, since the assumption of conditional independence of classes is not always respected.

Bayes theorem allows you to calculate the posterior probability:

$$P(c|x) = \frac{P(x|c)\,P(c)}{P(x)} \tag{3.9}$$

where $P(c|x)$ is the posterior probability of a given class c (that is, a given value of the target variable) for a given value of the attribute x.

$P(c)$ is the a priori probability of this class.

$P(x|c)$ is plausibility, i.e. probability of a given attribute value for a given class.

$P(x)$ is the a priori probability of a given characteristic value.

Its meaning at the philistine level can be expressed as follows. *Bayes theorem* allows you to rearrange the cause and effect. Knowing with what probability the

cause leads to a certain event, this theorem allows us to calculate the probability that this particular cause led to the observed event.

The purpose of the classification is to understand what class the document belongs to, so we need not the probability itself, but the most likely class. The naive Bayesian classifier uses a posteriori maximum estimation to determine the most likely class. Roughly speaking, this is a class with the maximum probability.

$$c_{map} = \text{argmax}\frac{P(d|c)\,P(c)}{P(d)} \tag{3.10}$$

That is, we need to calculate the probability for all classes and choose the class that has the maximum probability. Note that the denominator (document probability) is a constant and can in no way affect the ranking of classes, so in our problem we can ignore it.

$$c_{map} = \text{argmax}\,[P(d|c)\,P(c)] \tag{3.11}$$

Despite its simplicity, the *Naive Bayes classifier* can be surprisingly accurate. For example, it was found that it can be successfully applied to filter spam.

Implementation of the classifier. To implement the *naive Bayesian classifier*, we need a training set in which correspondences between text documents and their classes are indicated. Then we need to collect the following statistics from the sample that will be used at the classification stage:

- Relative frequencies of classes in the body of documents. That is, how often documents of this or that class meet
- The total number of words in the documents of each class
- Relative frequency of words within each class
- Sample dictionary size. The number of unique words in the selection.

Creating a system for analyzing opinions is a difficult task, but quite feasible if there is data for training and a domain (topic) is predetermined. When using machine learning, it is important to test different parameters in order to select those that work better on test data. In particular, it is necessary to test different classification algorithms (NB, SVM), a set of features (unigrams, bigrams, symbolic N-grams), a feature weighting function. There are still a lot of ways

to improve the classification of tonality, such as the use of tonal dictionaries, additional linguistic features (for example, parts of speech), and general ways to improve machine learning (boosting, bugging, etc.).

3.4 Types of Sentiment Analysis

Sentiment analysis is performed at several logical levels. At the *document level*, the task is to classify the document's tonality: whether it is positive or negative, sometimes a neutral case is also considered. At the *level of the proposal*, the task is to classify the mood of the individual sentence: positive or negative. It is also possible to consider the 3rd case – a neutral mood. And finally, the *level of aspects or entities*, the task is to classify the emotional coloring of individual sentences or phrases for certain entities or aspects.

3.4.1 Document Level Sentiment Analysis

The purpose of a *document level sentiment analysis* is to predict the overall polarity expressed in the document. In documents for which this type of analysis is applied, as a rule, the author assesses only one object (for example, reviews of products or services, hotels, books, films, etc). The task of predicting tonality at the document level can be viewed as a standard text classification problem. Then it can be solved using machine learning methods such as *logistic regression*, *naive Bayes classifier*, etc. There are several assumptions related to the approach to text classification. First, it is assumed. that the entire text is devoted to a single purpose, namely, the object that is the subject of review. Secondly, the author is considered the owner, the holder of opinion.

Formally, the task of determining tonality at the document level is defined as follows. There are many documents D with the opinions of writers, the classification determines what opinion, positive or negative, about the object contains each document $d \in D$. The document d is a review of the object O, which can be a product, a person, an organization, a thing, etc, O is depicted as a hierarchical tree of components and subcomponents. Each node represents a component and a set of attributes associated with it.

An opinion can be made about any node or attribute of a node. The term aspect or attribute is used to designate components and their attributes. Each

object *oo* is represented as a finite set of features $F = \{f_1, f_2, \ldots, f_n\}$, in which, as a special attribute, the object itself is included. Each sign $f_i \in F$ can be expressed by one phrase or a word from a finite set $W = \{w_{i_1}, w_{i_2}, \ldots, w_{i_n}\}$, which are synonyms of this aspect.

The opinion on the object O in the document d is expressed regarding a subset of its features F, is represented by a five

$$(O, f_i, OO_i, h, t) \tag{3.12}$$

where f_i is a feature of the object, OO_i is the tonality of opinion with respect to the feature f_i of the object O, h – owner of the opinion , t is the time at which the opinion was expressed, while h, t, O are considered known or irrelevant.

Such a definition is valid for customer feedback on products and services, with the so-called direct opinion, but may not be suitable for a forum or blog if the author writes about several products using comparative sentences – comparative or relative opinion.

3.4.2 Sentence Level Sentiment Analysis

The task of predicting the tonality or polarity of a sentence is also solved by classification. Formally, it is defined as follows: sentences S is given, and two subtasks need to be completed:

- Classification of subjectivity: determine whether S subjective suggestion or objective
- Classification of tonality at the sentence level: if is subjective, determine whether it expresses a positive or negative opinion.

The task of predicting tonality at the sentence level can also be solved using machine learning methods.

3.4.3 Aspect or Entity Level Sentiment Analysis

At the aspect level, the task of analyzing the tonality is as follows. First, feature-aspects are highlighted that have been commented on, for example, "This camera has an amazing image quality", a sign "image quality". Then it is determined

whether the opinion on this aspect is positive or negative, you can add a neutral case. For class prediction, all information about an entity is collected by its attributes, for example, "amazing" for the entity discussed above. Thus, on the basis of the attributes of an aspect-entity, the polarity of each aspect is determined. This task was formalized as a fine-grained sentiment analysis or a detailed analysis of tonality, here aspects are called essence of opinion, in addition to highlighting the opinion and the object of opinion directly, it also takes into account its owner, the relationship between them. There are three types of opinion entities:

- Opinion expressions or direct subjective expression – expressions that clearly define emotions, feelings, opinions
- Opinion targets – targets indicate what this opinion is about
- Opinion holders – from whom or from what opinion originates.

In the example below, the expressions of opinion (O) are underlined, the goals (T) and the owners of opinion (H) are enclosed in square brackets:

[Students] [$H_{1,2}$] were annoyed with [O_1] [uncertainty in curriculum] [T_1] and irritated [O_2] went to the next pair.

Numeric indices denote relationships. It should be noted that the goals of the opinion may be "empty", in connection 2 the emotion "irritated" is clearly not related to anything.

3.4.4 Review of Existing Solutions

Currently, there is already a fairly large number of ready-made systems for analyzing tonalities and seeking opinions. We give some of them.

- *Stanford NLP* – an open demo model of Stanford University, which allows to determine the tonality for film reviews. The system is based on the use of recursive neural networks. Supports texts exclusively in English.
- *Sentiment140* is a solution developed by Stanford alumni. It positions itself as the analyzer of the tonalities of the Twitter microblogging system. Allows the user to receive, in response to his request, a selection of positive, negative or neutral microposts. Renders the corresponding result using infographics. The service works only in English and Spanish.

- *30dB* is a free platform. Similar to the previous service, it accepts a request for input and gives out emotional opinions regarding the topic received. As the data for analysis uses not only Twitter, but also Facebook, Google+. As an additional option allows you to compare the emotive component of the two entered topics at once. Supports only English.

- *VAAL* – domestic development, developed since 1992. The system allows you to predict the effect of unconscious effects of texts on a mass audience, analyze texts from the point of view of such effects, identify the personality and psychological qualities of the authors of the text, carry out emotional-lexical and content analysis of texts, and produce automatic text categorization.

3.5 Natural Language Processing

Natural language processing (NLP) is one of the directions of artificial intelligence and mathematical linguistics, which is engaged in the study of problems of computer analysis and synthesis of natural languages.

We decipher some concepts that are part of this definition.

Natural language is a complex system of rules stored in the mind of a person, according to which speech activity occurs, i.e. generation and understanding of texts.

Artificial intelligence is a field of computer science (computer science), specializing in the simulation of human intellectual and sensory abilities with the help of computing devices.

Mathematical linguistics is a mathematical discipline, the subject of which is the development and study of concepts that form the basis of a formal apparatus for describing the structure of natural languages (that is, the metalanguage of linguistics).

Analysis as the form and method of the activity of artificial intelligence is interpreted as an understanding of the language. By *synthesis*, in the field of artificial intelligence, they mean the generation of literate text.

3.5.1 Basic Natural Language Processing Tasks

Understanding, recognition of natural language is a key task, since recognition and recognition of the language of the living requires the immense knowledge of the language system, language structure, their characteristics and patterns.

Define the main, most relevant tasks of natural language processing:

- First of all, consider one of the most important tasks, namely *speech recognition.* By this process is meant a process leading to the transformation of the speech signal of a human voice into digital information. Such an opportunity can be used by people who are deprived of the ability to type using hands or to simplify and speed up this process.

- *Speech synthesis* – the formation of speech signals in the printed text, that is, the artificial production of human speech. This branch of modern computer science, computational linguistics, and information technology, such as artificial intelligence, is engaged in this task. This task is mainly intended for use in information and reference systems, dispatch services, for issuing information requests about technological processes, to help people with impaired vision and speech.

- *Text analysis* is the process of extracting meaningful, high-quality information from natural language text to automate the process of extracting and analyzing data.

- We defined the *text synthesis* in the previous paragraph of this chapter. It has some specific subtasks. For example, multilingual generation. So called the automatic preparation of special documents in several languages (patent formulas, operating instructions for technical products or software systems). To solve this type of problem, detailed language models are used.

- *Machine or automatic translation.* Under this task of natural language processing is meant the process of translating oral texts written in natural language into another, also natural, language using electronic computers in computer programs intended for this type of tasks.

- *Creation of question-answer systems*, that is, such information systems that are capable of receiving, recognizing, classifying questions and giving an-

swers to them in natural language. This task is carried out according to the following algorithm:

- Determining the type of issue;
- Search for texts potentially containing an answer to this question;
- Extract the answer from these sources.

Such systems can be classified into:

- those that are designed to work with texts and topics of a particular subject;
- those that are able to work with information relating to different areas of knowledge.

- *Information search* – the process of identifying information in documents contained in databases accessible by the search system that correspond to a given query on a topic. The execution of this task implies the execution of the following sequence of operations:
  - formulation of an information request;
  - search for potential holders of relevant information;
  - extracting information from found documents;
  - familiarization with the results of the search and selection of the most suitable sources for the query conditions.

  There are 3 types of information retrieval:
  - search by content of the entire document;
  - search by document name, creation date, author, size, etc. data;
  - search in subject of the image, a subject which is present at it.

- *Extracting information* – NLP tasks that automatically extract the necessary data from a source of information, text (usually unstructured). This task is directly related to the previous one.

- *Analysis of the tonality of the text* – analysis of the lexemes of the text, evaluation of their emotional coloring and classification by belonging to a neutral, positive or negative lexical layer of the language.
- *Referencing* – reducing the amount of text due to the selection of the main theses by searching for matches given in the search keywords and its summary.

4. Results and Discussion

This section presents the use of various methods of machine learning to solve problems associated with the processing of natural language and the results of experimental studies for the analysis of tonality and their description. One of these problems is the automatic determination of the emotional coloring (positive, negative, neutral) of textual data, that is, analysis of tonality. The purpose of this task is to determine whether a given text is positive, negative or neutral in its effect on the reputation of a particular object. The difficulty of analyzing the tonality lies in the presence of an emotionally enriched language – slang, polysemy, uncertainty, sarcasm, all these factors are misleading not only people, but also computers.

Before any data processing requires preprocessing (data cleaning). In this stage, all html tags, punctuation, symbols are removed. This operation is performed using the python library - "Beautiful Soup". Also, all numbers and links in the text are replaced by tags. Further in the text there are so-called "stopwords" – these are private words in the language, which basically do not carry any meaningful meaning (for example, in English these are words like "the", "at", "about", etc.). Stopwords are removed using the *Python Natural Language Toolkit (NLTK)* package. As a consequence, a consistent step is the conversion of words into vectors. The fact is that the computer, as well as mathematical formulas, is easier to work with numbers, and not a set of words. Therefore, we need to present any text as a vector of numbers. To do this, you can make a dictionary with all the words, i.e. merge all the words found in the texts into one large dictionary and replace the words from the text with the index in the dictionary. There are several types of word embedding methods, including *word2vec* and *fastText*, which are most popular for representing words. This study uses simple tokenization in neural networks and the most popular *word2vec* word embedding algorithm. Finally,

Self-Generating Neural Network architectures are applied to the learning model, then they are saved and used to predict new text to test performance.

In general, the training model may have a different number of levels, but in this study it consists of four levels, which generally include the level of implementation, the level of RNN, two dense layers with different input characteristics. For the entire study, we divided the main data set into three sets: training, cross-checking and testing. Since the cross-validation kit plays a major role in obtaining more reliable classifier results, the model was trained using the 20-fold cross-validation kit. In addition, Adam's optimization algorithm and binary cross-entropy loss are used for the optimizer and the loss function, respectively. For activation, sigmoidal functions and ReLU functions are used.

For learning the model, not only optimization or loss functions are important, but also the packet size and the number of epochs, where one epoch is when the entire data set is transmitted back and forth to the neural network only once, and the lot size is the total number of training examples. present in one lot. The relationship between these two concepts is that one era is divided into smaller lots because it is too large to be calculated. However, the correct number of epochs and packet size are unknown, and it can vary from domain to domain, from a larger data set to a smaller data set. Typically, the most commonly used batch size is 20, 25, and 32. In addition, it is important to use the cross-check set as verification data in the learning model and mix the data set.

This study uses a fixed number of epochs and batch size: 5 and 20, respectively. The reason for choosing a small number of epochs is to avoid retrofitting. More precisely, retrofitting is caused even when using only 5 epochs in both data sets. The reconfiguration in the neural network appears when the model fits well with the training data; however, in the new invisible data or verification data they do not classify the data correctly.

In other words, the model studied and memorized patterns specific to training data that are not related to validation data. In practice, retrofitting is identified by examining the loss or accuracy of the test and comparing it with the loss or accuracy of training. As a rule, the loss of training continues to decrease, while the loss of testing, on the contrary, began to increase after several epochs. To avoid retrofitting or to solve this problem, there are several ways, such as early stop, regular dropout, traditional regularization, reducing the size of the model,

getting more data, and normalizing packets. In this study, methods are used to overcome migration, with the exception of the last two methods: obtaining more data and normalizing the batch. As mentioned earlier, a small number of epochs are used, and the model is small. In addition, in the following subsections, the results of experiments with and without regularization methods are shown and compared.

In this thesis, an accessible data set was used for Kazakh texts [5], with the addition of my work and library (in our case approximately 20015 reviews, of which 7002 are positive, 6945 are negative and 6068 are neutral). Evaluation of posts was carried out on a scale from -1 to 1.

The choice of an evaluation metric is changed depends on the task. For NLP tasks, which include sentiment analysis, used various types like accuracy(a), precision(p), recall(r) and measure(F). The mathematical formulas of them are:

$$a = \frac{tp + tn}{tp + tn + fp + fn}, \tag{4.1}$$

$$p = \frac{tp}{tp + fp}, \tag{4.2}$$

$$r = \frac{tp}{tp + fn}, \tag{4.3}$$

where true positive (tp) – a positive real value is correctly classified as positive, false positive (fp) – a negative real value is classified as positive, true negative (tn) – a negative real value is correctly classified as negative, false negative (fn) – a positive real value is classified as negative.

So,

$$F = \frac{2 * p * r}{p + r}, \tag{4.4}$$

The programming language Python was used to implement proposed models in this work since it has a large number of scientific libraries for data processing and machine learning algorithms. To perform machine learning or deep learning tasks, there exist several tools or libraries such as *TensorFlow*, *Theano*, *Scikit-learn* , *Keras* and etc.

Keras is an open source Python library that makes it easy to create neural networks. The library is compatible with *TensorFlow*, *Microsoft Cognitive Toolkit*,

Theano and *MXNet*. *Tensorflow* and *Theano* are the most commonly used numerical Python platforms for developing deep learning algorithms, but they are quite difficult to use. *Keras*, on the other hand, provides a simple and convenient way to create deep learning models.

Keras library has been selected for this thesis since it is so well-suited to the concepts of neural networks and building of simple or complex neural networks takes a few minutes by the help of some powerful models, such as the Sequential model and the Model class used with the functional API. While for building neural networks part in the research used *Keras* framework, then for word representation part used Python package *Gensim* in order to handle words to vectors. It is a great package for processing texts and working with word vector models such as *word2vec* and *fastText*.

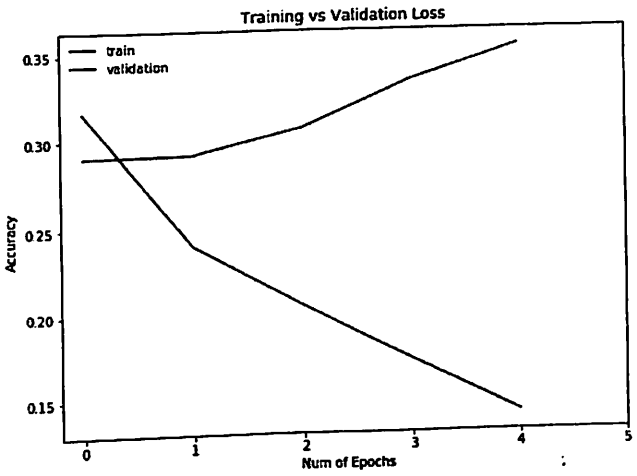
4.1 Results for Experiments with Tokenizer

This specific section only presents the results for experiments that are done by using simple tokenizer under neural networks.

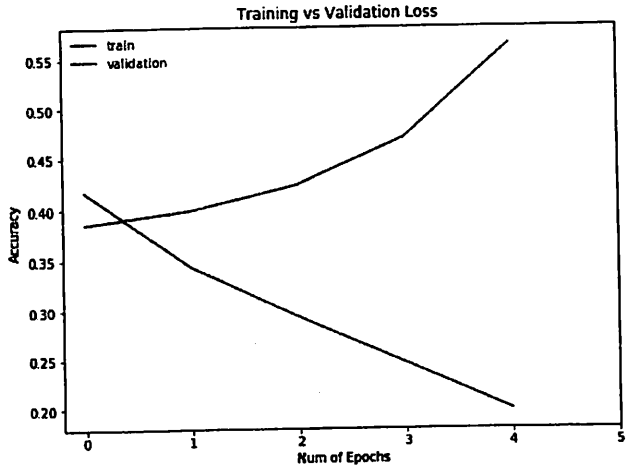
Tokenizer allows to vectorize a text corpus, by turning each text into either a sequence of words ids or into a vector where the coefficient for each token could be binary, based on word count, and based on frequency. Keras provides the Tokenizer class for preparing text documents for deep learning. However, simple Tokenizer has been constructed by ourselves for this research and has been used to tokenize Kazakh sentences.

As shown in 4.1, the overfitting problem occurrs on dataset when applying, both RNN models, LSTM, and bidirectional LSTM. To address the issue dropout regularization and most popular L2 regularization are applied. Despite the fact that the dropout regularization is the powerful tool to address overfitting, in our case it was not the solution to the problem. On the other hand, the L2 regularization technique is the alternative way to solve the problem and to avoid overfitting from the training model. The overfitting problem is addressed using L2 regularization and can be seen in the 4.2.

4.1 presents the results of 20-fold cross-validation of different models on the training set and the results on the test set of a Dataset. The metrics used are the accuracy and F1-score by classes of positive and negative polarities.

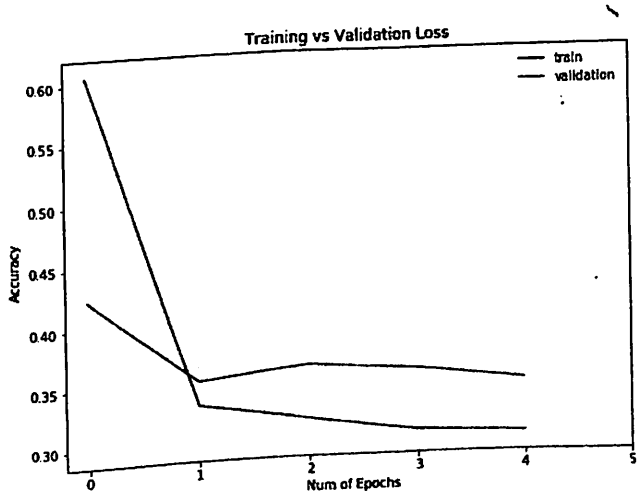


(a) LSTM model

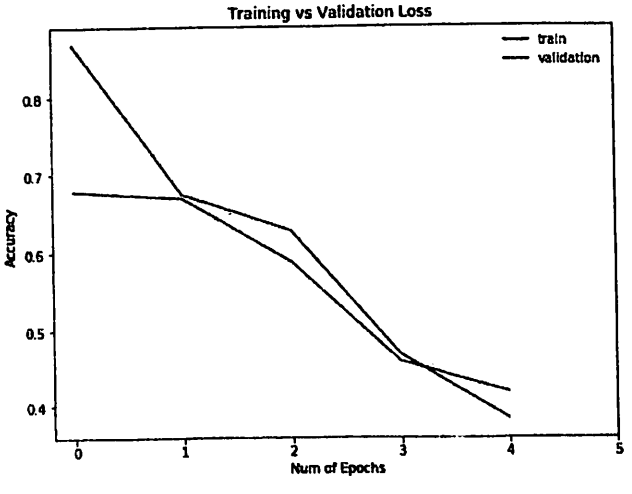


(b) Bidirectional LSTM

Figure 4.1: An example of overfitting.



(a) LSTM model with L2 regularization



(b) Bidirectional LSTM with L2 regularization

Figure 4.2: An example of avoided overfitting.

Table 4.1: The results for Dataset

DATASET	Validationa data Accuracy	Test data F1-score
LSTM	87.3	87.1
LSTM with Dropout	87.4	87.6
LSTM with L2 regularization	87.8	88.4
Bidirectional LSTM	87	87.3
Bidirectional LSTM with Dropout	87.3	87.3
Bidirectional LSTM with L2 regularization	87.9	88

Since the overfitting is the biggest issue during whole research, the results of the models with L2 regularization are taken as more correctly performance for the datasets. So, LSTM and bidirectional LSTM with L2 regularization show 88.4% and 88% as their F1-score for Dataset. Each of the architectures gives acceptable results for Kazakh datasets by using simple tokenizer as a vector representation of words.

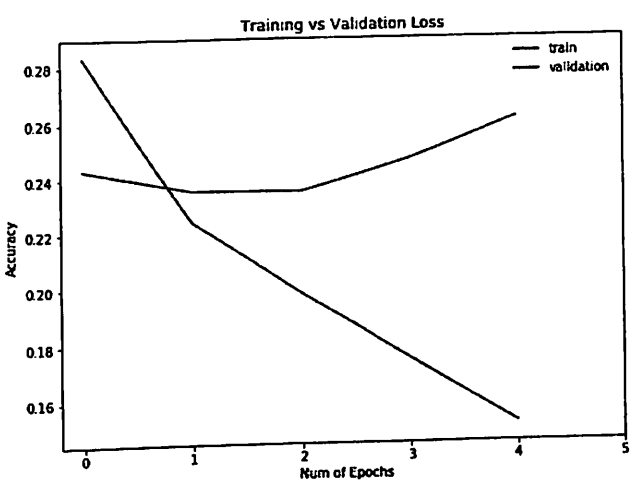
4.2 Results for Experiments with Word2Vec

This section only presents the results for experiments that are done by using state-of-the-art word embedding algorithm, so called word2vec. For this research, a pre-trained word2vec model for Kazakh language is utilized.

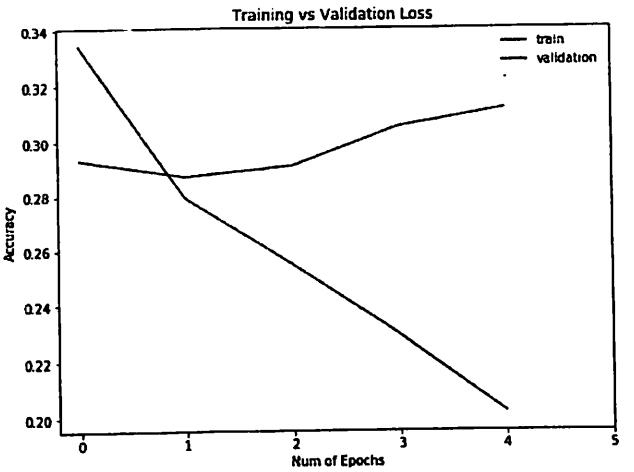
It can be seen from 4.3 that overfitting starts at some point instead of starting at the beginning point as in the previous section. In such situation to avoid the overfitting, the early stopping method can be used with the epoch where the validation loss starts increasing. However, in this research the regularization methods are also applied to avoid overfitting. The difference from the previous section when used simple tokenizer under neural networks is validation loss, fluctuated when dropout regularization is applied to overcome the overfitting (see 4.4). Finally, the L2 regularization method avoids overfitting from the training model shown in 4.5.

The results for experiments using Word2Vec embedding for Dataset can be seen in 4.2. The accuracy and F1-score metrics are used for binary classification.

From the table, it can be seen that depending on the dataset and the model the

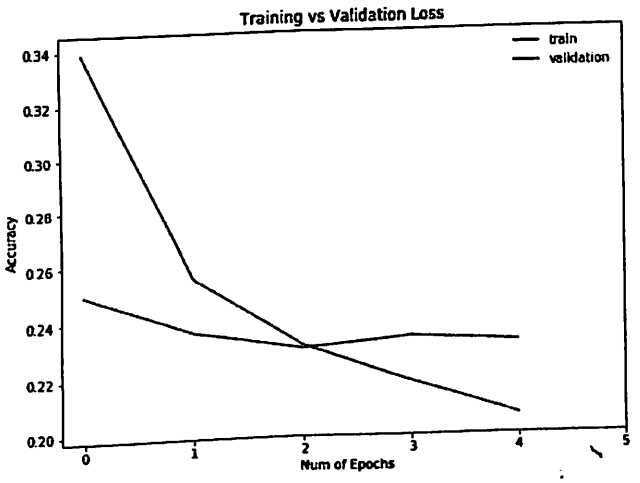


(a) LSTM model

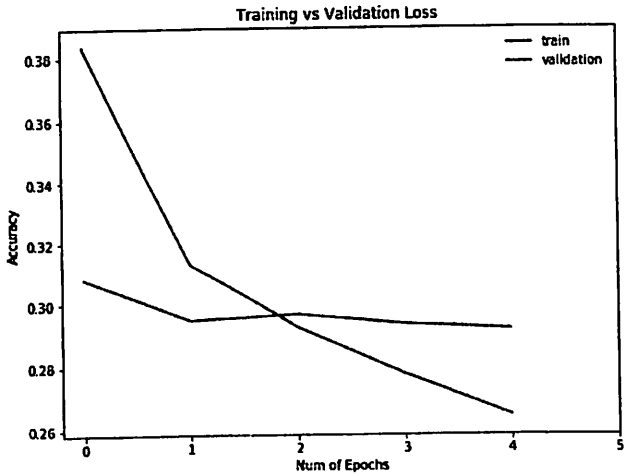


(b) Bidirectional LSTM

Figure 4.3: An example of overfitting.

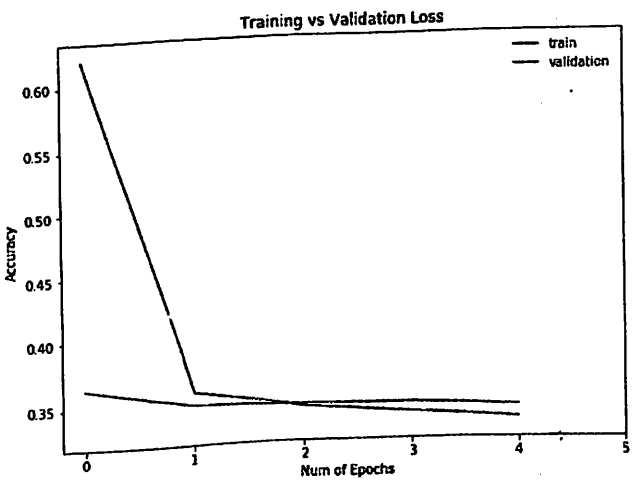


(a) LSTM model with dropout regularization

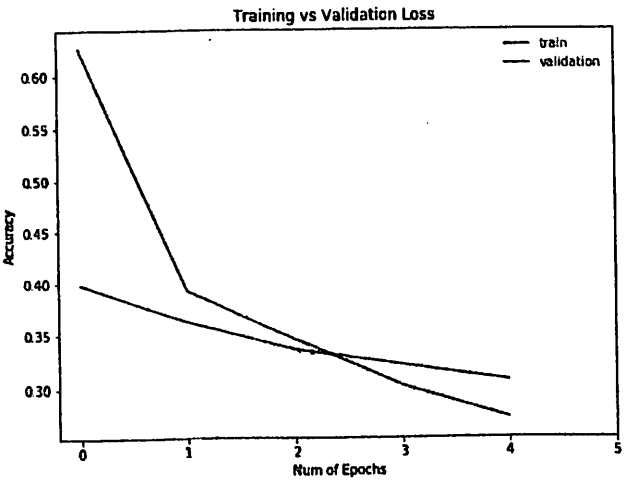


(b) Bidirectional LSTM with dropout regularization

Figure 4.4: An example of fluctuated overfitting.



(a) LSTM model with L2 regularization



(b) Bidirectional LSTM with L2 regularization

Figure 4.5: An example of avoided overfitting.

Table 4.2: The results for Dataset

DATASET	Validationa data Accuracy	Test data F1-score
LSTM	89.5	84.3
LSTM with Dropout	90.2	85.1
LSTM with L2 regularization	87.5	81.2
Bidirectional LSTM	89.2	83.8
Bidirectional LSTM with Dropout	90	85.1
Bidirectional LSTM with L2 regularization	89.2	83.8

predicted test can be far worth than the validation or training data. Since overfitting problem cannot be avoided by using dropout regularization, the outcomes made by using L2 regularization would be taken as a result. LSTM architecture gives 81.2% F1-score for Dataset, while for the same dataset bidirectional LSTM provides performances of 83.8% for F1-score. To conclude, the bidirectional LSTM overcomes the simple LSTM model in most cases.

5. Conclusion

In this thesis, the *Self-Generating Neural Networks* are classified for *Sentiment Analysis* task on Kazakh texts (in our case approximately 20015 reviews, of which 7002 are positive, 6945 are negative and 6068 are neutral). The main concept of using *Self-Generating Neural Networks* models for this sentiment classification problem is the lack of researches using them for Kazakh language. Generally, their architectures provide excellent results for English language and this research encourages this fact since it achieves reasonably good results for Kazakh language, too. Furthermore, word representations play a key role in classifying sentiment analysis tasks and choosing the right word representations are also important to achieve reasonably good results.

In the first part of research, the simple *tokenizer* under neural network is used as a word representation and presents more superior results than the word embedding model *word2vec*. In general, *word2vec* model performs outstanding results for English texts but in this research it achieves reduced results as compared to the simple *tokenizer* for the same dataset. It can be explained by the lower capacity of pre-trained Kazakhs *word2vec* model. In spite of this, both word representations with *Self-Generating Neural Networks* models present results between 81% and 88% for our Dataset, which is preferably results for Kazakh domains.

Despite the fact that the models perform well, there can be done more research on changing and reducing the size of the model and taking another batch size to train the model should be carried out for further studies. Moreover, it is known that there exist several types of *loss functions*, *optimizations algorithms* which can be adapted and the vocabulary size which can be shaped to a smaller dimension. Therefore, all these changes tend to do more research and achieve valuable results for *Sentiment Analysis* tasks on Kazakh review domains.

As future work, the new, well-balanced dataset with larger size must be created

in order to train deep learning models more precisely for Kazakh reviews domain due to the fact that for deep learning models the extension of larger dataset is a significant one. Another approaches different from *Self-Generating Neural Networks* can be implemented to solve sentiment analysis on Kazakh texts. More precisely, architectures such as *Convolutional Neural Networks* (CNN), *fastText* for text classification, *Attention Mechanism* and *Recursive Neural Networks* can be applied since there no works related to these approaches in Kazakh.

Different from the approaches mentioned earlier, the word representation is also another important part for improving the performance of models. So, one of the most popular text representation methods is *Universal Sentence Encoder* representations and its various types. In recent years, these methods applied with deep leaning models have become more active in the area of *Natural Language Processing*. Moreover, another approach is the so called transfer learning method which is used earlier for Kazakh domains and highly improved the performance of baseline models.

On the whole, since the *Document Level* is not just one type of *Sentiment Analysis* then the other levels, sentence and aspect-based, can be a new research work for *Self-Generating Neural Network* architectures.

6 REFERENCES

[1] Jakovlevs S., *Perceptron Architecture Ensuring Pattern Description Compactness* // Scientific Journal of Riga Technical University, Vol. 40, p. 87-93

[2] http://www.statsoft.com/textbook/statistics-glossary/l

[3] Titov E., Simonov V., Erpelev A., *Classes on Artificial Intelligence, Robotics and Automation with Students of Technical Specialties* // Proceeding of the 24th Conference of Fruct Association

[4] Gorbachev S., Abramova T., *Intelligent Multidimensional Data and Image Processing*

[5] https://github.com/chapayevdauren/sentiment-analysis-for-kz