TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology

Nikolai Jefimov 143689IASM

IMAGE RECOGNITION BY SPIKING NEURAL NETWORKS

Master's thesis

Supervisor: Eduard Petlenkov
PhD

Tallinn 2017

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Nikolai Jefimov 143689IASM

KUJUTISE TUVASTAMINE IMPULSI NÄRVIVÕRKUDEGA

Magistritöö

Juhendaja: Eduard Petlenkov
PhD

Tallinn 2017

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Nikolai Jefimov

30.05.2017

Abstract

The aim of this thesis is to present opportunities on implementations of spiking neural network for image recognition and proving learning algorithms in MATLAB environment.

In this thesis provided examples of the use of common types of artificial neural networks for image recognition and classification. Also presented spiking neural network areas of use, as well as the principles of image recognition by this type of neural network.

The result of this work is the realization of the learning algorithm in MATLAB environment, as well as the ability to recognize handwritten numbers from the MNIST database. Besides this presented the opportunity to recognizing Latin letters with dimensions of 3x3 and 5x7 pixels.

In addition, were proposed User Guide for masters course ISS0023 Intelligent Control System in TUT.

This thesis is written in English and is 62 pages long, including 6 chapters, 61 figures and 4 tables.

Annotatsioon

Kujutise tuvastamine impulsi närvivõrkudega

Selle väitekiri eesmärgiks on tutvustada impulsi närvivõrgu kasutamisvõimalused, samuti näidata õpetamise algoritmi MATLAB keskkonnas.

Töös esitatakse levinumaid tehisnärvivõrgu kasutamise näidet kujutise tuvastamiseks ja klassifitseerimiseks. Esitatud impulsi närvivõrkude kasutamise alad ja kujundite tuvastamise põhimõtted.

Magistritöö lõpptulemusena on rakendatud õpetamise algoritm MATLAB keskkonnas, lisaks on võimalik tuvastada käekirja numbrit MNIST andmebaasist. Peale selle on võimalik tunnustada ladina tähestiku suurusega 5x7 ja 3x3 pikslit.

Välja on töötatud õpematerjal/ praktilise töö juhend magistriõppe aine ISS0023 Arukad juhtimissüsteemid jaoks.

Lõputöö on kirjutatud Inglise keeles ning sisaldab teksti 62 leheküljel, 6 peatükki, 61 joonist, 4 tabelit.

List of abbreviations and terms

ANN	Artificial neural network
FFNN	FeedForward neural network
SLNN	Self-Learning neural network
RBFN	Radial based function network
RNN	Recurrent neural network
HTM	Hierarchical temporal memory
SNN	Spiking neural network
FPGA	Field-programmable gate array
nPU	Neural processing unit
IBM	International Business Machines
CMOS	Complementary metal-oxide-semiconductor
STDP-learning	Spike-timing-dependent plasticity
LIF	Leaky integrate-and-fire
jAER	Java tools for Address-Event Representation
DVS	Dynamic Vision Sensor
AER-EAR	Address-event representation ear
DBN	Deep belief network
RBM	Restricted Boltzmann machine
MNIST	Mixed National Institute of Standards and Technology

Table of contents

1. Artificial neural networks 12
1.1 Feedforward neural network 13
1.2 Radial basis function network 14
1.3 Recurrent neural network 15
1.4 Physical neural network 16
1.5 Other types of network 17
1.5.1 Neuro-fuzzy networks 17
1.5.2 Hierarchical temporal memory 18
1.5.3 Spiking neural networks 19
1.6 Comparison 20
2. Spiking neural network as a tool 21
2.1 Spiking models 21
2.2 Application where used spiking neural networks 22
2.2.1 Prosthetics 23
2.2.2 Robotics 24
2.2.3 Hardware 25
2.2.4 Computer vision 26
3. Handwriting processing in MATLAB with spiking neural network 28
3.1 Methodology 28
3.1.1 Deep Belief Neural Network and Restricted Boltzmann Machines 28
3.1.2 Discrete-time and event-driven neuron models 30
3.2 Training the network 31
3.2.1 Input data 31
3.2.2 Network Architecture 32
3.2.3 Training 33
3.3 Result 33
4. Minimal recognition size 36
4.1 Dataset 36
4.2 Setup 36
4.3 Result 37

5. Character recognition 39
5.1 Task 39
5.2 Training results with FeedForward Neural Network 40
5.3 Training with Self-Learning Neural Network 41
5.4 Training with Spiking Neural Network 44
5.5 Comparison of the results 46
6. Summary 47
References 48
User guide 53

List of figures

Fig. 1.1 *Human neuron; artificial neuron; biological synapse; ANN synapses* 12
Fig. 1.2 *Structure of feedforward neural network* 13
Fig. 1.3 *Feedforward neural network recognition demo* 13
Fig. 1.4 *Feedforward neural network recognition workflow* 13
Fig. 1.5 *Structure of radial basis function* 14
Fig. 1.6 *Example of area categorizing using RBFN* 14
Fig. 1.7 *The architecture of a recurrent neural network* 15
Fig. 1.8 *RNN in combine with convolutional neural network can be used* 15
Fig. 1.9 *Physical neural network liquid state machine utilizing nanotechnology* 16
Fig. 1.10 *Physical robots and neural network controller* 16
Fig. 1.11 *Example of Takagi-Sugeno rules. Equivalent Neuro-fuzzy network* 17
Fig. 1.12 *Emotion recognition with neuro-fuzzy logic* 17
Fig. 1.13 *HTM network with four levels* 18
Fig. 1.14 *Visualization of estimated results with HTM recognition* 18
Fig. 1.15 *Example of "brain-like" model of spiking neural network* 19
Fig. 2.1 *Principal components of SNN-based neuroprosthetic control paradigm* 23
Fig. 2.2 *BrainOS build-in module in cleaning machine* 24
Fig. 2.3 *Qualcomm's brain-inspired chip* 25
Fig. 2.4 *IBM Neuromorphic System* 26
Fig. 2.5 *Video feed recognition with TrueNorth chip* 27
Fig. 3.1 *Boltzmann and Restricted Boltzmann Machines* 28
Fig. 3.2 *Example of MNIST and MATLAB dataset sample (7)* 32
Fig. 3.3 *Spiking DBN architecture for image recognition* 32
Fig. 3.4 *Training result for MNIST dataset recognition* 33
Fig. 3.5 *RBM weights for MNIST recognition* 34
Fig. 3.6 *Recognition of MNIST sample 48 (7)* 34
Fig. 3.7 *Recognition of MNIST sample 655 (1)* 34
Fig. 3.8 *Recognition of MNIST sample 6592 (4)* 35
Fig. 3.9 *Recognition of MNIST sample 7362 (0)* 35
Fig. 3.10 *Recognition of MNIST sample 74 (9)* 35
Fig. 3.11 *Recognition of MNIST sample 492 (8)* 35
Fig. 4.1 *Fakoo alphabet* 36

Fig. 4.2 *Training result of Fakoo alphabet recognition* 37
Fig. 4.3 *Recognition of Fakoo alphabet sample 3 (C)* 37
Fig. 4.4 *Recognition of Fakoo alphabet sample 15 (O)* 37
Fig. 4.5 *Recognition of Fakoo alphabet sample 20 (T)* 38
Fig. 4.6 *Recognition of Fakoo alphabet sample 24 (X)* 38
Fig. 5.1 *Example of characters A, B and C as matrix* 39
Fig. 5.2 *Example of characters A, B and C as image* 39
Fig. 5.3 *Letter A with noise 20%, 30% and 22%* 39
Fig. 5.4 *FFNN lab task accuracy result* 40
Fig. 5.5 *FFNN lab task time result* 40
Fig. 5.6 *FFNN lab task epoch* 40
Fig. 5.7 *Lab task testing data sample 4 (D)* 40
Fig. 5.8 *Lab task testing data sample 15 (O)* 40
Fig. 5.9 *Lab task self-learning verification with 10 epoch* 41
Fig. 5.10 *Lab task self-learning verification with 20 epoch* 41
Fig. 5.11 *Lab task self-learning verification with 25 epoch* 42
Fig. 5.12 *Lab task self-learning verification with 30 epoch* 42
Fig. 5.13 *SLNN lab task accuracy result* 43
Fig. 5.14 *SLNN lab task time result* 43
Fig. 5.15 *SLNN lab task epoch* 43
Fig. 5.16 *Lab task testing data sample 3 (C)* 43
Fig. 5.17 *Lab task testing data sample 17 (Q)* 43
Fig. 5.18 *SNN lab task accuracy result* 44
Fig. 5.19 *SNN lab task time result* 44
Fig. 5.20 *SNN lab task epoch* 44
Fig. 5.21 *Recognition of sample 3 (C)* 45
Fig. 5.22 *Recognition of sample 4 (D)* 45
Fig. 5.23 *Recognition of sample 15 (O)* 45
Fig. 5.24 *Recognition of sample 17 (Q)* 45

List of tables

Table 1.1 *Comparison of usage different ANN for image processing* 20
Table 4.1 *SNN setup for Fakoo alphabet recognition* 36
Table 5.1 *Lab task self-learning neural network recognition verification* 41
Table 5.2 *Lab task result comparison* 46

1. Artificial neural networks

Artificial neural network is a software and/or hardware realization of a mathematical model, based on principles of organization and functioning of the biological neural network.

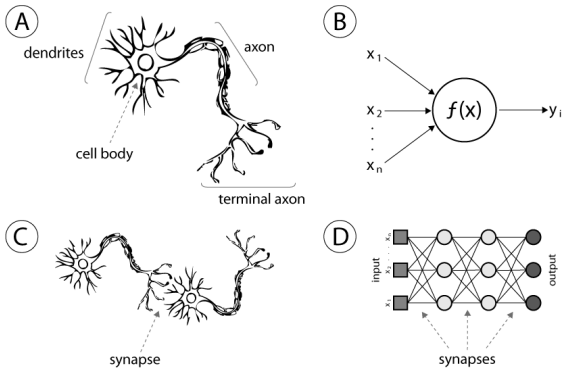


Fig. 1.1*(A) Human neuron; (B) artificial neuron; (C) biological synapse; (D) ANN synapses [1]*

Last generation neural network usually operates up to several million artificial neurons and axons. But this size is not enough to fully reproduce human brain and can be compared to thinking level of a protozoa [2].

To "teach" neural network can be used one of the learning technics:

- Supervised learning – for each case provided situation and requested solution
- Unsupervised learning – provided only situation, agent categorize results itself
- Reinforcement learning – network cooperate with "environment" without knowledge about system

In general, ANN can be classified in a groups, which are listed and explained below.

Additionally, provided an example of image processing/recognition with this network type.

1.1 Feedforward neural network

First and, basically, most simple type on ANN. Information goes only in one direction – forward. No cycles or loops in the network. Can have from 1 to N numbers of inputs and outputs. Also, from 0 to N hidden layers [3].

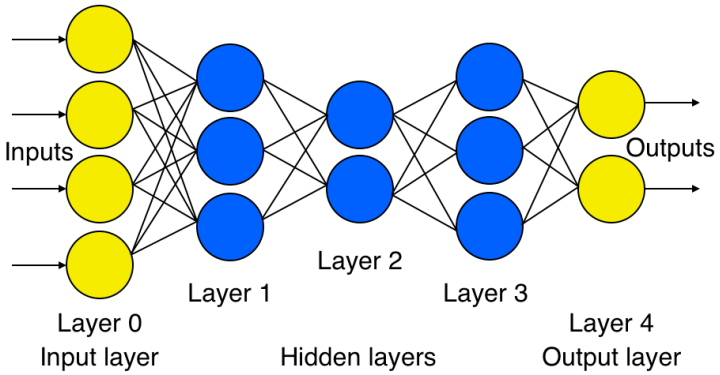


Fig. 1.2 *Structure of feedforward neural network*

This approach can be used for simple optical character recognition.

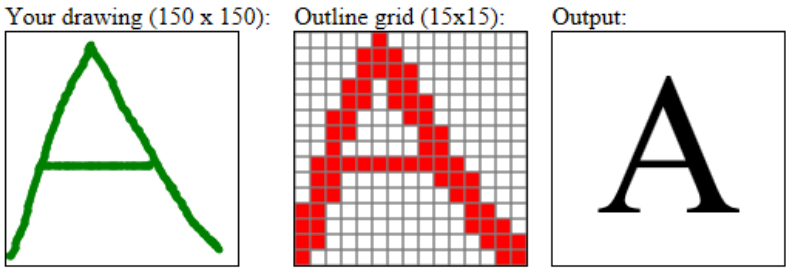


Fig. 1.3 *Feedforward neural network recognition demo [4]*

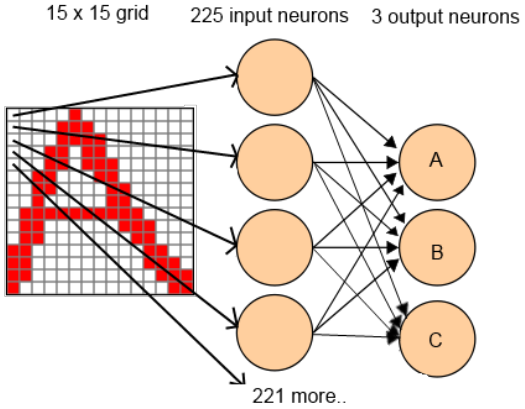


Fig. 1.4 *Feedforward neural network recognition workflow [4]*

1.2 Radial basis function network

RBFN have some advantages over FFNN [5]:

- This type of network models arbitrary nonlinear function only with one hidden layer. This helps to remove requirements in additional hidden layers
- The parameters of the linear combination in the output layer can be fully optimized using well-known linear optimization methods that work quickly.
- In addition, the RBFN network is trained very quickly.

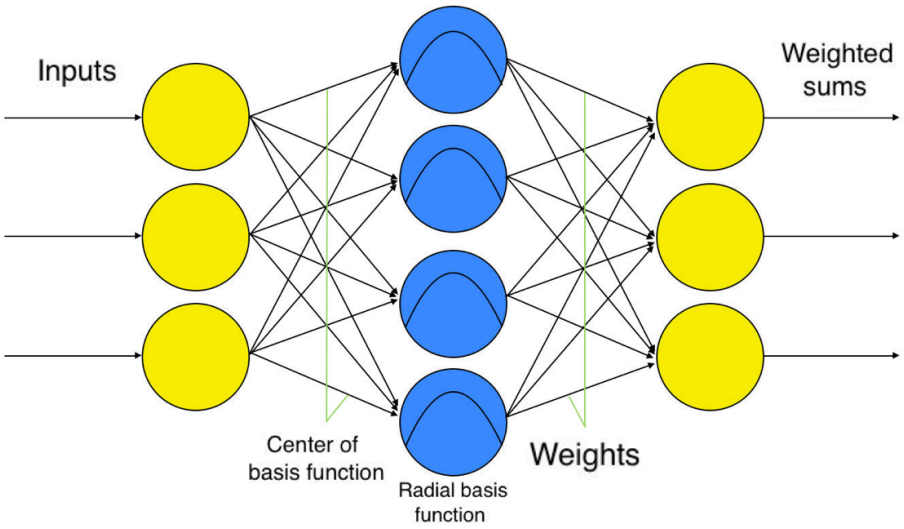


Fig. 1.5 *Structure of radial basis function*

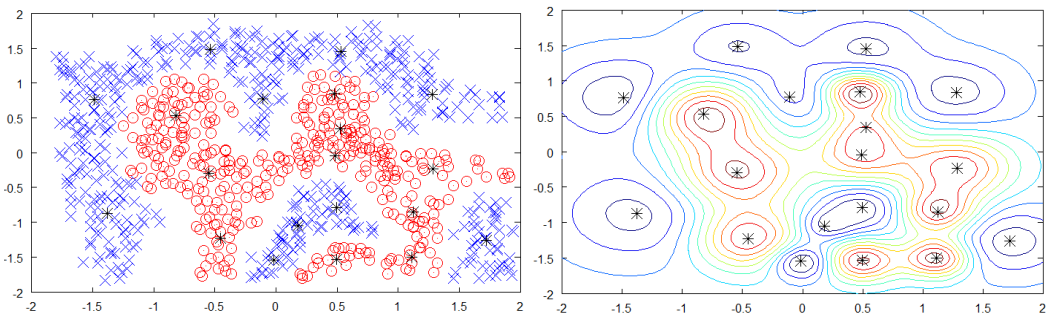


Fig. 1.6 *Example of area categorizing using RBFN [6]*

1.3 Recurrent neural network

The main point of the RNNs in an opportunity to use feedback. In the classical neural network, a signal flows only straight forward and this limits implementation field. Every single calculation had influence to the follows input Presence of feedback allows access to networks internal memory, which can help it field of recognition and classification problem [7].

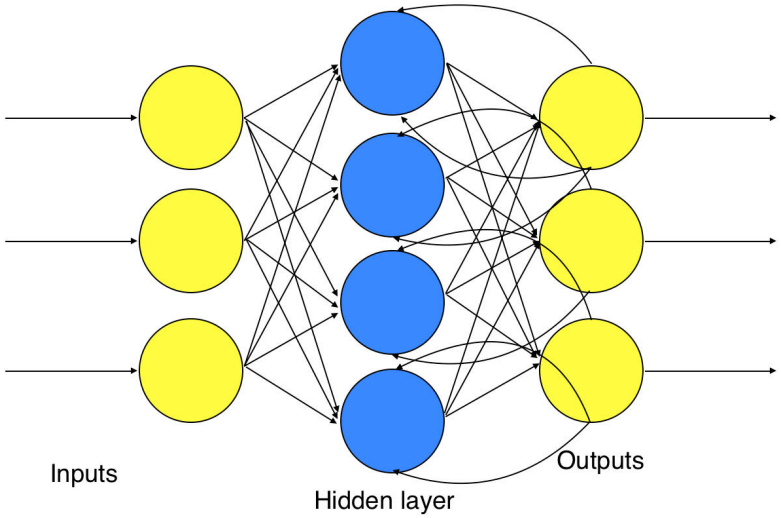


Fig. 1.7 *The architecture of a recurrent neural network*

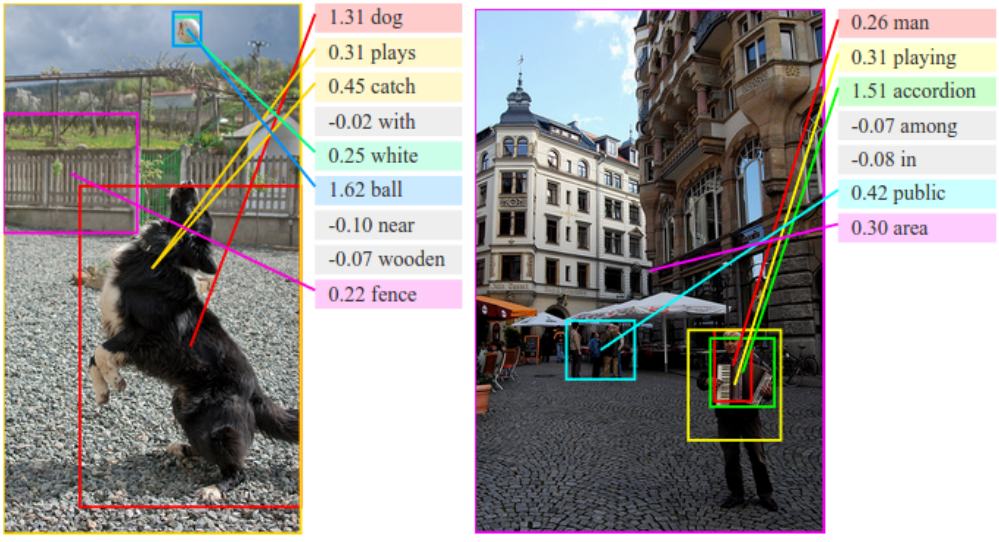


Fig. 1.8 *RNN in combine with convolutional neural network can be used for generation description of unlabeled images [8]*

1.4 Physical neural network

The main difference from “traditional” artificial neural networks is that in physical neural networks electrical components are used to imitate brain activity.

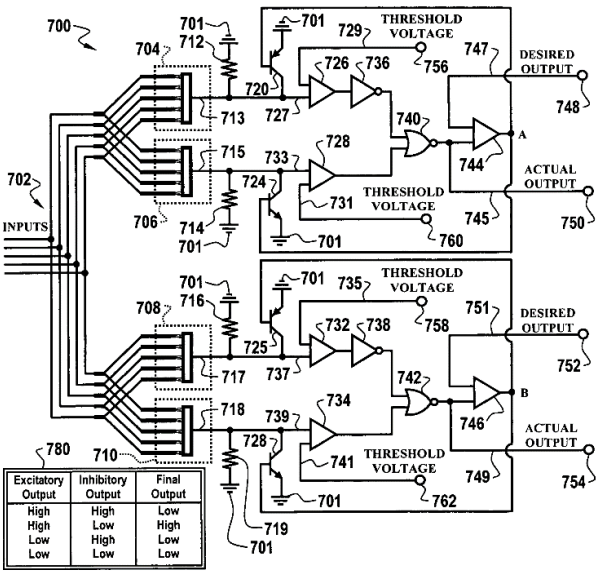


Fig. 1.9 *Physical neural network liquid state machine utilizing nanotechnology [9]*

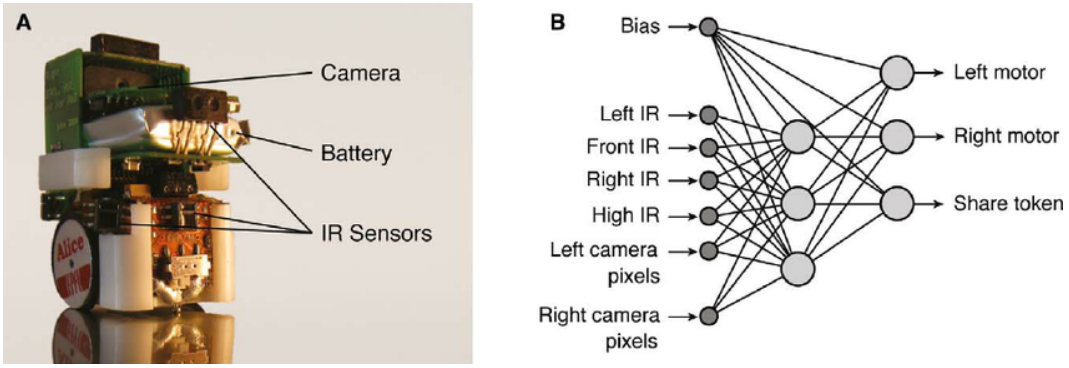


Fig. 1.10 *Physical robots and neural network controller [10]*

1.5 Other types of network

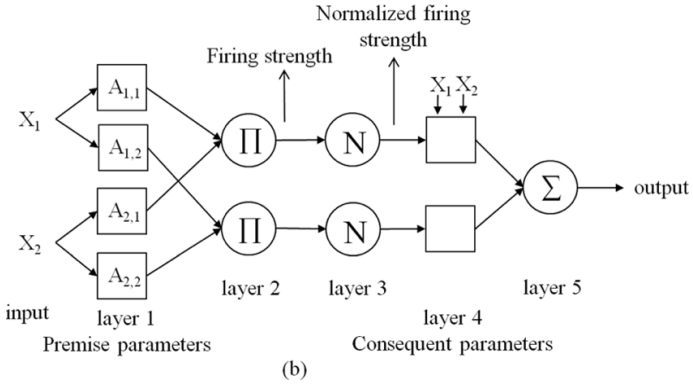
A few words about self-sustained neural networks

1.5.1 Neuro-fuzzy networks

From the network name it is clear, that this method based on the fuzzy logic implemented in ANN. Using this approach reveals opportunity to use neuro-fuzzy networks as universal approximators with focus to IF-THEN rules [11].

$$r_1 : IF\ X_1\ is\ A_{1,1}\ \ and\ X_2\ is\ A_{2,1}\ THEN\ y_1 = p_{1,0} + p_{1,1}X_1 + p_{1,2}X_2$$
$$r_2 : IF\ X_1\ is\ A_{1,2}\ and\ X_2\ is\ A_{2,2}\ THEN\ y_2 = p_{2,0} + p_{2,1}X_1 + p_{2,2}X_2$$

(a)



(b)

Fig. 1.11 *(a) Example of Takagi-Sugeno rules. (b) Equivalent Neuro-fuzzy network [11]*

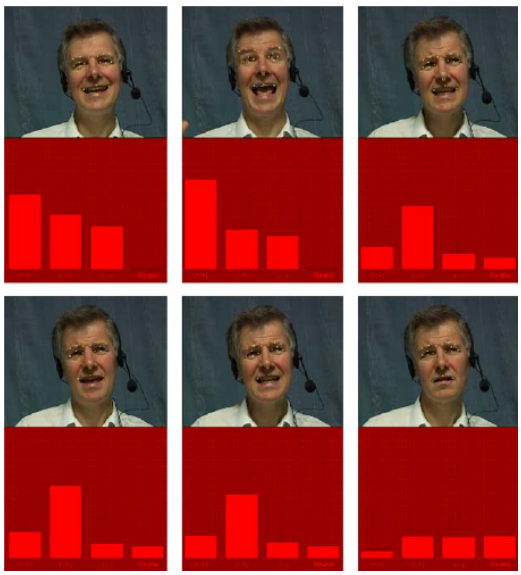


Fig. 1.12 *Emotion recognition with neuro-fuzzy logic [12]*

1.5.2 Hierarchical temporal memory

Hierarchical temporal memory is an unsupervised to semi-supervised online machine learning model, which models some of the structural and algorithmic properties of the neocortex. HTM is a biomimetic model based on the memory-prediction theory of brain function [13].

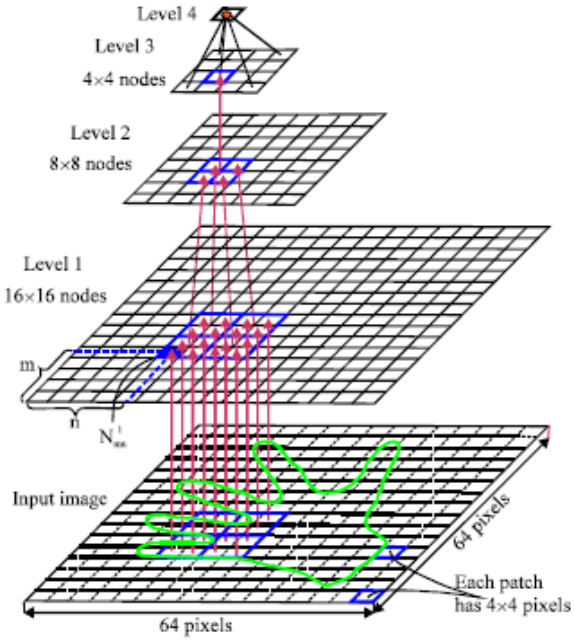


Fig. 1.13 *HTM network with four levels* [*14*]

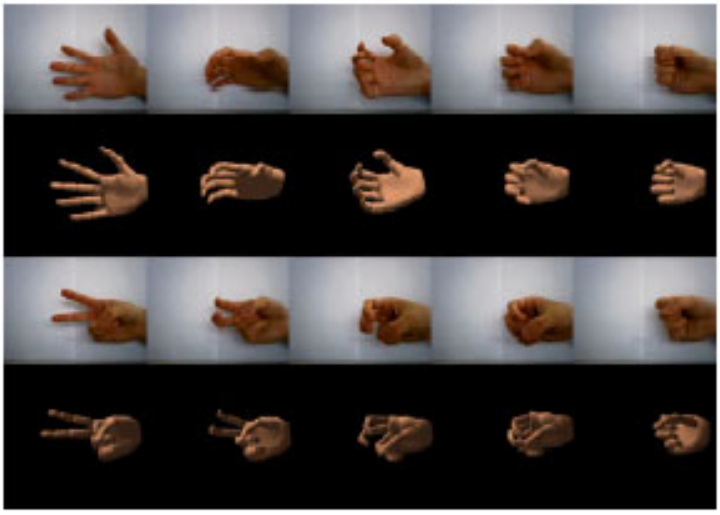


Fig. 1.14 *Visualization of estimated results with HTM recognition [14]*

1.5.3 Spiking neural networks

At the moment, SNNs are considered one of the most advanced neural networks. The reason is in the very close representation of the concept of the human brain. Besides synaptic and neuronal state, SNN has included an idea of time in the workflow. The point of the SNN is in the activity of the neurons. Unlike traditional multi-layer ANNs, in SNNs neurons is active only then the membrane potential, a difference in electric potential between neurons, is exceeded predetermined value. After neuron activation, generated spikes reach other neurons and cause a neurons potentials change in favor with received value (increase or decrease of potential) [15].

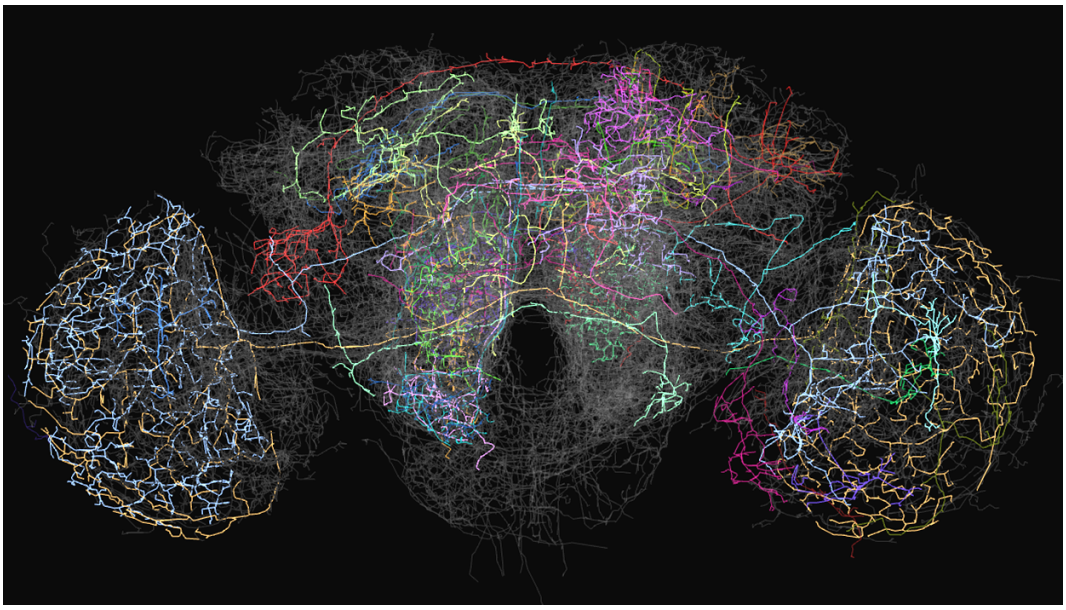


Fig. 1.15 *Example of "brain-like" model of spiking neural network [16]*

Opportunities of using spiking neural networks as the most advanced technic for image processing and recognition I will provide in the next chapter.

1.6 Comparison

Any of mentioned above ANNs can be used for image processing and recognition, but all of them have they own pros and cons.

	Field of use (most common)	**Complexity of images to process**	**Ability to learn**	**Computation power (similar task or close to it) in abstract units [17]**
Feedforward neural network	Recognition	Simple image with minimal noise and disturbance	Training Pattern	1
Radial basis function network	Classification	Must present similar pattern for classification	Training Pattern	0.7
Recurrent neural network	Classification	Must be combined with other NN	Training Memory	0.8
Physical neural network	Control	Live image	Training Pattern	0.5
Neuro-fuzzy network	Recognition	Live image	Training Pattern	1.3
Hierarchical temporary memory	Recognition	Live image with minimal disturbance	Training Memory	1.6
Spiking neural network	Recognition	Live image	Learn in process Prediction	0.9

Table 1.1 *Comparison of usage different ANN for image processing*

Based on the requirement assigned for neural network for image recognition, the best choice is spiking neural network. It can work with work with large live feed using less computational power.

2. Spiking neural network as a tool

As was mentioned before, spiking neural networks are the most advanced development in intention to replicate human brain. It was achieved by implementing individual spike. It's revealing opportunity to include spatial-temporal information in cross layer connection. Traditionally rate coding was used, now neurons use pulse coding. This means that neurons process individual pulses what is allows for information multiplexing. Research shows that this is how real neurons work [18].

2.1 Spiking models

As a model, spiking artificial neural network have the same internals as their biological analogs [19]:

- They can process information coming from many inputs and release single spike as response.
- The probability of spike generations is increased by excitatory inputs and decreased by inhibitory inputs
- When neuron activation threshold is achieved, spikes most be generated.

Spiking neural network models can be divided in three main categories [20]:

1. Feedforward networks – no feedback connection. Signal move only straight forward from input to output. Can be applied multilayer system.
2. Recurrent networks – neurons interact with each other not only forward, but also with feedback connection to the previous layer. This approach helps SNNs remonstrate dynamic temporal behavior.
3. Hybrid networks – in this category presented 2 types of SNNs in which have in structure feedforward and recurrent combined:

    - Synfire chain – A multi-layered architecture in which spike pulses can spread as a synchronous wave of neuron activation from one layer of the chain to other.
    - Reservoir computing – Abstraction paradigm, that uses opportunities of recurrent network, but at the same time don’t have training disadvantages.

2.2 Application where used spiking neural networks

The last decade was marked by the rapid development of the third generation of neural networks - spiking neural networks. This was caused by the emergence of the opportunity to develop and produce large hardware neural networks implemented in the form of chips, which made it possible to introduce a replacement in many areas of application of the traditional von Neumann architecture computers to a spiking neural network based solution with low energy consumption.

There are some advantages which have SNN over neural networks of previous generations [21]:

- SNNs are dynamic, and therefore perfectly suited for working with dynamic processes (speech recognition and dynamic images);
- SNNs have multitasking, because the input data is processed in a neural network with feedbacks, and different groups of reading neurons can be trained to solve different tasks;
- SNNs are able to perform foresight recognition (it is not necessary to have complete information about the object or to know the result of the process);
- SNN is simply to teach since it is sufficient to train only the output reading neurons;
- SNNs have increased information processing and noise immunity because they use a timeline information presentation;
- SNN requires a smaller number of neurons since each neuron of a pulsed neural network replaces two neurons (exciting and inhibitory) of the classical ANN;
- SNNs have a high speed of operation and a great potential for parallelization since for the transmission of the pulse it is necessary to send 1 bit, rather than a continuous value, as in frequency ANNs;
- SNN can be trained in the process of work.

Disadvantages are also presented:

- It is not advisable to use the SNNs in systems with a small number of neurons;
- There is no perfect learning algorithm.

Based on the opportunities of using SNNs, main directions of implementations can be highlighted.

2.2.1 Prosthetics

At the moment most developed field of implementation of SNNs in medicine. Especially in neuroprosthetics. Since the principles of coding sensory information entering the brain are known, appeared the idea to emulate the signals in diseased, injured or amputated sense organs and feed them through the electrodes to the nerves coming from these senses or even directly to regions of the cerebral cortex responsible for processing of relevant sensory information. Similarly, knowledge of the coding of commands coming from the brain to the muscles makes it possible for them to be interpreted by special prostheses controlled by microprocessors that reproduce the actions normally performed by a healthy limb [22].

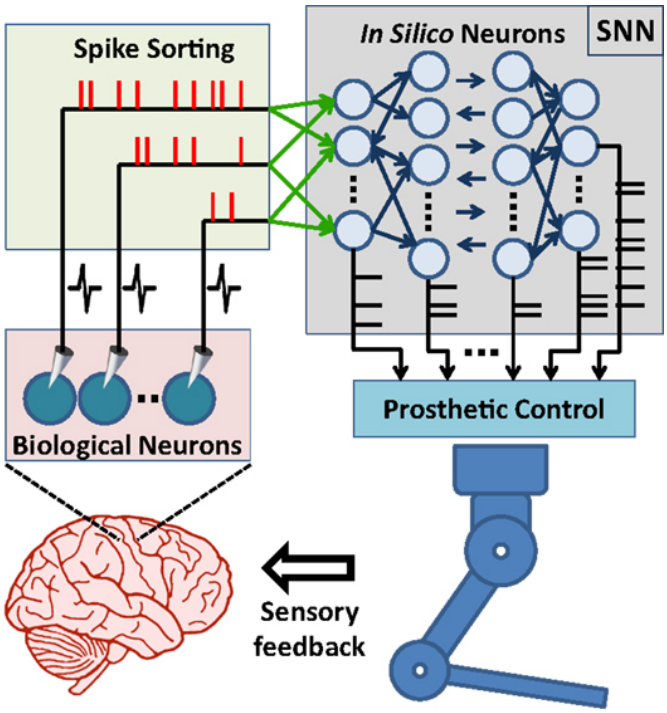


Fig. 2.1 *Principal components of SNN-based neuroprosthetic control paradigm [23]*

For over 10 years, there are visual neuroprostheses that provide signals from arrays of light-sensitive elements that are encoded as a series of spikes in the visual cortex departments completely blind patients, giving them the ability to navigate in space and even read. Another long and successful history of developing the use of hearing neuroprosthesis in patients with profound hearing loss (including the deaf from birth) [23].

Another area of neuroprosthetics is direct electrical stimulation of the brain. The sending of spike sequences through electrodes implanted in certain deep structures of the brain helps to alleviate or completely eliminate the symptoms of Parkinson's disease, dystonia, chronic pain and even psychiatric diseases (manic-depressive psychosis, schizophrenia). Here we are talking about directed intervention in the brain.

2.2.2 Robotics

Since spiking neural networks have ability to “see” and analyze they can be used as a part of advanced robotics. Here is needed to mention a company named “Brain Corporation” from San-Diego, USA. Founder of this company is Dr. Eugene Izhikevich. He is known for his foremost of the theory of spiking networks. Dr. Izhikevich with his team implemented the world’s largest thalamo-cortical model. “Brain Corporation” is software company specializing in the development of intelligent, autonomous systems that automate commercial equipment. The company is now focused on developing advanced machine learning and computer vision systems for the next generation of self-driving vehicles [24].

Fig. 2.2 *BrainOS build-in module in cleaning machine [24]*

2.2.3 Hardware

At the moment 2 major player in the field of computing power are working on they own so-called “neural processing unit”.

“Intel is interested in this because use of this approach can accelerate specific functions (e.g. complex neural networks, video codecs or search algorithms) and could deliver up to 10x performance efficiency across a variety of workloads, and integrating the FPGA with coherent and non-coherent links within the Xeon package (versus discreet FPGAs) could lead to an additional 2x performance improvement.” [25]

Qualcomm have they own research in neurocomputing. Advantages of Zeroth processor is in the ability to recreate the way of brain behavior. Zeroth are created for image and sound processing. It means that it can recognize and analyze faces, gestures and also speech. Besides that, it can be used for optimizing processes, for example for battery life extension [26].

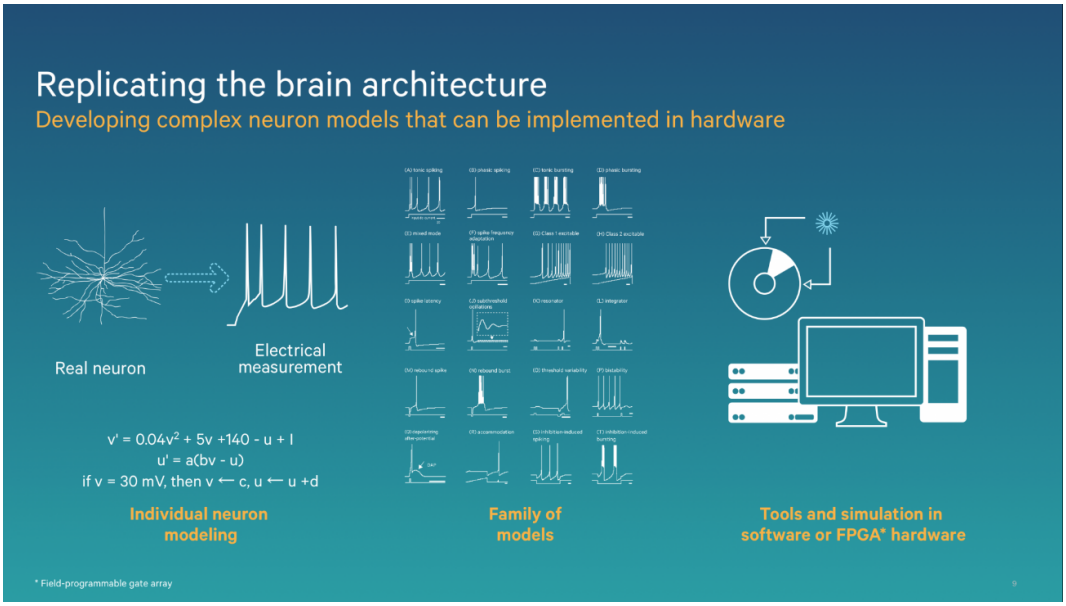


Fig. 2.3 *Qualcomm's brain-inspired chip [26]*

2.2.4 Computer vision

SNN have been successfully used for image classification. They provide a model for the mammalian visual cortex, image segmentation and pattern recognition. Different spiking neuron mathematical models exist, but their computational complexity makes them ill-suited for hardware implementation at the moment [27].

But still, IBM has developed neuromorphic chip called “True North”. “True North consists of 1 million programmable neurons and 256 million programmable synapses conveying signals between the digital neurons. With a total of 5.4 billion transistors, the computer chip is one of the largest CMOS chips ever built. Yet it uses just 70 mW in operation and has a power density about 1/10,000 that of most modern microprocessors. That brings neuromorphic engineering closer to the human brain’s marvelous efficiency as a grapefruit-size organ that consumes just 20 W.” [28]

Applying this approach can eliminate von Neumann computing restriction, when several tasks cannot be performed in one processing unit in the same time.

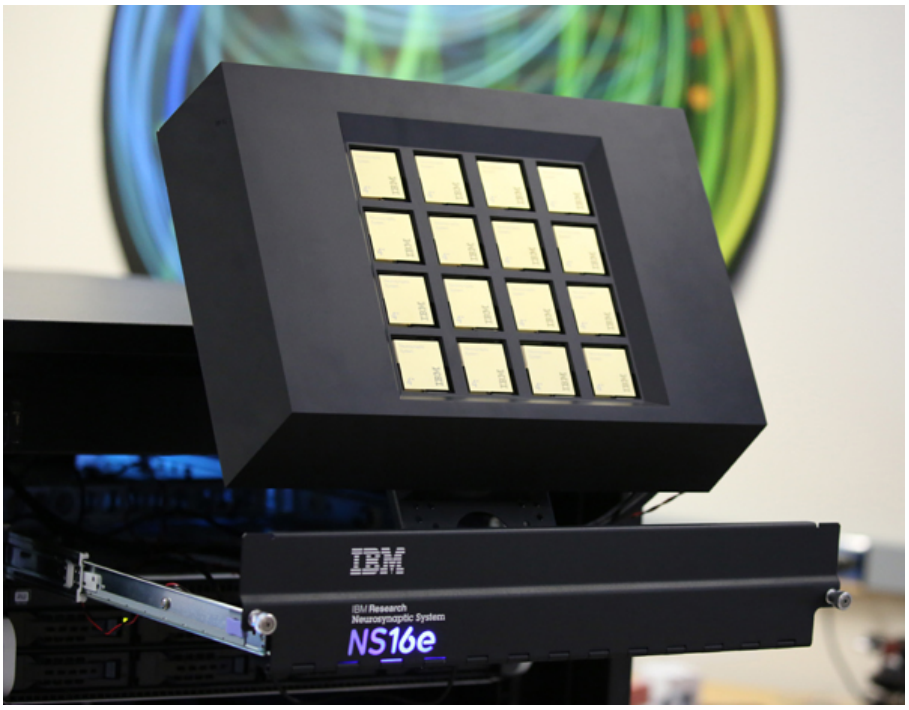


Fig. 2.4 *IBM Neuromorphic System [29]*

Fig. 2.5 *Video feed recognition with TrueNorth chip [30]*

"A video camera on Hoover Tower at Stanford University is looking down at the plaza, below. A simulated network of IBM TrueNorth chips takes in the video data and locates interesting objects. Objects might look interesting to the system because they are moving or have a different color or texture than the background. The system then further processes those portions of the interesting video to determine what the objects are. It is trained in several specific categories, such as buses, cars, people, and cyclists. In a monitoring application, the camera would only need to communicate when it found an interesting object, rather than continually streaming video to a central location." [30]

3. Handwriting processing in MATLAB with spiking neural network

As an example of image processing, I will consider handwriting recognition solution developed by research team from Institute of Neuroinformatics, University of Zurich and ETH Zurich, Zurich, Switzerland. This method is based on the jAER open-source project and training process is implemented in MATLAB environment [31].

This project was developed as proof-of-concept for real time recognition of handwritten digits and letters from 128x128 Dynamic Vision Sensor with pre training in MATLAB. I will focus only in the part of recognition and result analysis for MATLAB environment.

3.1 Methodology

3.1.1 Deep Belief Neural Network and Restricted Boltzmann Machines

DBN can be described as graphical model with several hidden layers. This means that neurons within one layer are not connected with each other, but connected with neurons from other layers. RBM are using stochastic structure approach but based on the same principles. Connections inside this model are only between “visible” and “hidden” group nodes, not within one group. In comparison with Standard Boltzmann Machine in Restricted model nodes of one layer are not connected. This makes learning and classification process tractable in RBM [32].By stacking RBM in form of a DBN, the lower layer of RBM become of the visible layer of the higher RBM. This structure allows teach RBM to analyze more specific tasks [33].

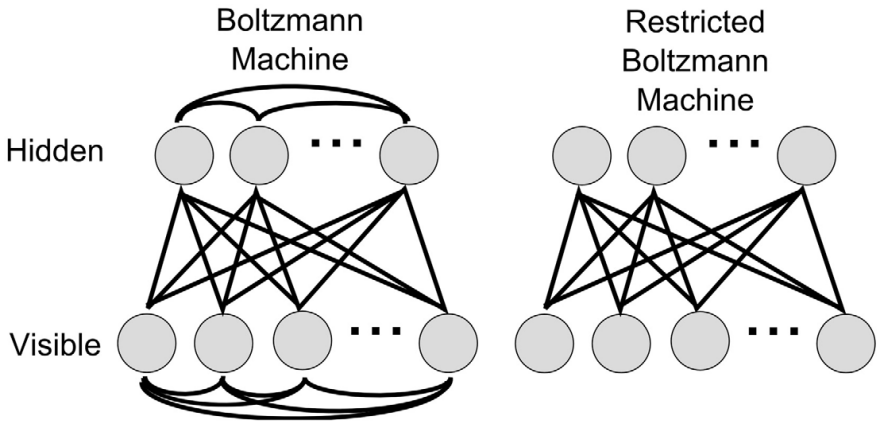


Fig. 3.1 *Boltzmann and Restricted Boltzmann Machines [31]*

"In a binary RBM the units stochastically take on states 0 or 1, depending on their inputs from the other layer. Denoting the states of visible units with v_i, the states of hidden units with h_j, the weights connecting these units with w_{ij}, and the biases of visible and hidden units with $b_i^{(v)}$ and $b_j^{(h)}$ respectively, a RBM encodes a joint probability distribution $p(\mathbf{v}, \mathbf{h}|\theta)$, defined via the energy function" [31]

$$E(v,h;\theta) = -\sum_i \sum_j w_{ij}\, v_i h_j - \sum_i b_i^{(v)}\, v_i - \sum_j b_j^{(h)}\, h_j, \tag{1}$$

where $\theta = (\mathbf{w}, \mathbf{b}^{(v)}, \mathbf{b}^{(h)})$. The encoded joint probability can then be written as

$$p(v,h|\theta) = \frac{\exp\left(-E(v,h;\theta)\right)}{\sum_{v'} \sum_{h'} \exp\left(-E(v',h';\theta)\right)}. \tag{2}$$

"From equations (1) and (2) the following stochastic update rules for the states of units were derived, such that on average every update results in a lower energy state, and ultimately settles into an equilibrium [34]:" [31]

$$3p(v_i = 1|h,\theta) = \sigma\left(\sum_j w_{ij}\, h_j + b_i^{(v)}\right) \tag{3}$$

$$4p\left(h_j = 1\middle|v,\theta\right) = \sigma\left(\sum_i w_{ij}\, v_i + b_j^{(h)}\right), \tag{4}$$

"where $\sigma(x) = 1/(1 + \exp(-x))$ is the sigmoid function, and the units will switch to state 0 otherwise. When left to run freely, the network will generate samples over all possible states ($\mathbf{v}$, $\mathbf{h}$) according to the joint probability distribution in (2). This holds for any arbitrary initial state of the network, given that the network has enough time to become approximately independent of the initial conditions." [31]

Previously was mentioned that we creating DBN by stacking RBM. This approach allows us to transform hidden layer of the RBM to the visible layer of next level. The upper levels of RBM will handle more abstract functions, which provides improved classification parameters [35].

3.1.2 Discrete-time and event-driven neuron models

"In the standard formulation, units within RBMs are binary, and states are sampled according to the sigmoidal activation probabilities from Equations (3) and (4). Such neuron are called models sigmoid-binary units. It was shown that an equivalent threshold-linear model can be formulated, in which zero-mean Gaussian noise N (0, σ^2_n) with variance σ^2_n is added to the activation functions [36]:" [31]

$$h_j = \max\left(0, \sum_i w_{ij} v_i + b_j^h + \mathrm{N}(0, \sigma_n^2)\right), \tag{5}$$

"In this model, each incoming event adds to the membrane potential *Vm* according to the strength w_{ij} of the synapse along which the event occurred. Incoming spikes within an absolute refractory period t_{ref} after an output spike are ignored. Spikes are generated deterministically whenever the membrane potential crosses the firing threshold V_{th}, otherwise, the membrane potential decays exponentially with time constant τ. Simple versions of LIF neurons can be simulated in an event-based way since membrane potentials only need to be updated upon the arrival of input spikes, and spikes can only be created at the times of such input events. For a LIF neuron representing h_j, which receives a constant input current $s_j = \sum_i w_{ij} v_i$ corresponding to the weighted *i* sum of inputs from connected visible units, the expected firing rate $\rho_j(s_j)$ is [37]:" [31]

$$\rho_j(s_j) = \begin{cases} \left(t_{ref} - \tau \log\left(1 - \frac{V_{th}}{s_j}\right)\right)^{-1} & if\ s_j \geq V_{th} \\ 0 & otherwise \end{cases} \tag{6}$$

"The above equation holds when the neuron is injected with a constant input, but under realistic conditions, the neuron receives a continuous stream of input spike trains, each arriving to first approximation as samples from a Poisson process with some underlying firing rate. For this case, a more accurate prediction of the average firing rate can be obtained using Siegert neurons [38]. Siegert neurons have transfer functions that are mathematically equivalent to the input-rate output-rate transfer functions of LIF neurons with Poisson- process inputs. In order to compute the Siegert transformation for a neuron receiving excitatory and inhibitory inputs with rates $(\vec{p}_e, \vec{p}_i)$ and weights $(\vec{w}_e, \vec{w}_i)$ respectively, we first have to compute the auxiliary variables

$$\mu_Q = \tau\sum(\overrightarrow{w_e}\overrightarrow{\rho_e} + \overrightarrow{w_i}\overrightarrow{\rho_i}) \quad (7) \qquad \Upsilon = V_{rest} + \mu_Q \quad (8)$$

$$k = \sqrt{\tau_{syn}/\tau} \quad (9) \qquad \sigma_Q^2 = \frac{\tau}{2}\sum\left(\overrightarrow{w_e^2}\overrightarrow{\rho_e} + \overrightarrow{w_i^2}\overrightarrow{\rho_i}\right) \quad (10)$$

$$\Gamma = \sigma_Q \quad (11) \qquad \gamma = \left|\zeta\left(\frac{1}{2}\right)\right| \quad (12)$$

where τ_{syn} is the synaptic time constant (for our purposes considered to be zero), and ζ is the Riemann zeta function. Then the average firing rate ρ_{out} of the neuron with resting potential V_{rest} and reset potential V_{reset} can be computed as [39]:" [31]

$$\rho_{out} = (t_{ref} + \frac{\tau}{\Gamma}\sqrt{\frac{\pi}{2}} \cdot \int_{V_{reset}+k\gamma\Gamma}^{V_{th}+k\gamma\Gamma} \exp[\frac{(u-\Upsilon)^2}{2\Gamma^2}] \cdot [1 + erf(\frac{u-\Upsilon}{\Gamma\sqrt{2}})]du)^{-1} \quad (13)$$

"A RBM trained using Siegert units can thus be easily converted into an equivalent network of spiking LIF neurons: By normalizing the firing rate in Equation (13) relative to the maximum firing rate $1/t_{ref}$, ρ_{out} can be converted into activation probabilities as required to sample RBM units in Equations (3, 4) during standard Contrastive Divergence learning with continuous units. After learning, the parameters and weights are retained, but instead of sampling every time step, the units generate Poisson spike trains with rates computed by the Siegert formula Equation (13)." [31]

3.2 Training the network

The source code for MATLAB can be found in the repository of developers [40]. From this point, a solution will be considered from the theoretical point of view. More detailed implementation can be observed in User Guide at the end of this work.

Point of this chapter is to evaluate recognition capability of this approach and understand, can it be used for other recognition and classification tasks.

3.2.1 Input data

As was mentioned previously, for understanding of opportunities of this solution will be used MNIST dataset. MNIST is a database of handwritten digits. By itself consists of 60000 training samples and 10000 test samples. Unfortunately, original data is stored in a format, which is not suitable for MATLAB.

We will use mnist_uint8.mat which is taken from Deep Learning Toolbox for MATLAB[41]. This is converted MNIST database for usage in MATLAB environment. In original MNIST input image is 28x28 pixels, but here it is modified as one vector input layer with 784 visual input units.

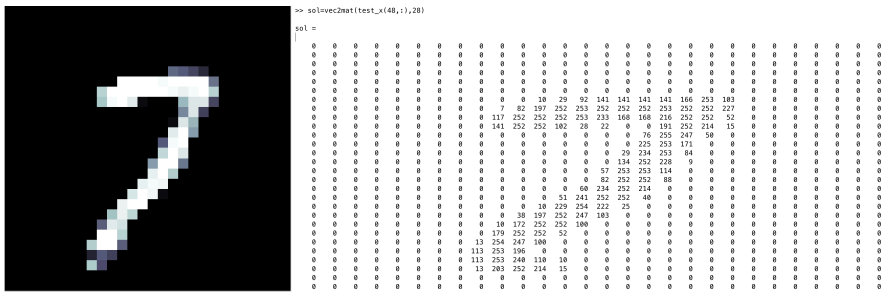


Fig. 3.2 *Example of MNIST and MATLAB dataset sample (7)*

3.2.2 Network Architecture

Because DBN was created by stacking RBMs, we can consider this network as "traditional". In the simplified view, it will look like:

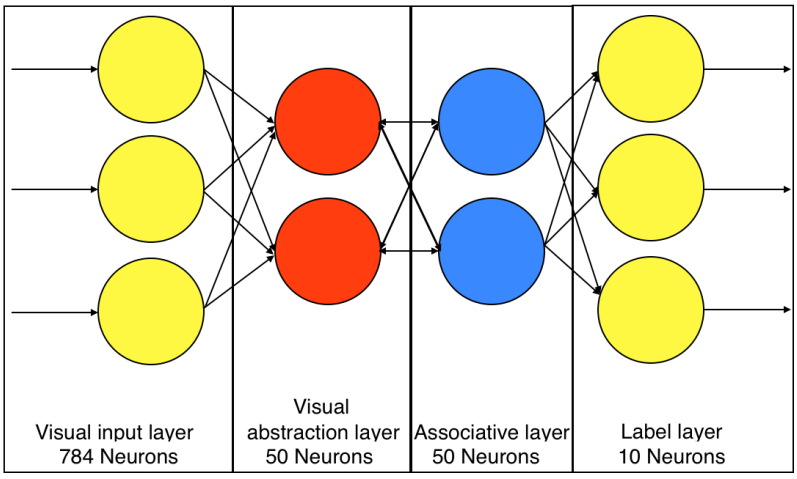


Fig. 3.3 *Spiking DBN architecture for image recognition*

Visual input layer with 784 neurons because the of input vector, equal 28x28 pixel dataset image.

Visual abstraction and Associative layers have both 50 neurons, this is not an optimal or perfect amount, but for understanding of opportunities, this is enough.

Label layer have 10 neurons in favor with digits from 0 to 9.

3.2.3 Training

Spiking neural network training process divided into a few steps. Since our model consists of many RBM, they should be trained individually.

- As an independent RBM, determinates weight coefficients within Input and Abstract layers.
- After that established supervised learning between Associative and Label layer. But as input in Associative layer used previously trained RBM (Input and Abstract)
- Every RBM must be trained predefined epoch times

3.3 Result

After the execution example.m with parameters from above, was achieved recognition rate 90.46%.

```
Beginning training.
Epoch 1: mean error: 0.01022.
Epoch 2: mean error: 0.00590.
Epoch 3: mean error: 0.00696.
Epoch 4: mean error: 0.00751.
Epoch 5: mean error: 0.00792.
Epoch 6: mean error: 0.00771.
Epoch 7: mean error: 0.00669.
Epoch 8: mean error: 0.00728.
Epoch 1: mean error: 0.01828.
Epoch 2: mean error: 0.02374.
Epoch 3: mean error: 0.02194.
Epoch 4: mean error: 0.02812.
Epoch 5: mean error: 0.01681.
Epoch 6: mean error: 0.01904.
Epoch 7: mean error: 0.02138.
Epoch 8: mean error: 0.02125.
Scored: 90.46
```

Fig. 3.4 *Training result for MNIST dataset recognition*

In *fig 3.5* we can see the distribution of weight coefficient in Abstract layer. Every single square represents a set of weight inside a neuron after performing a learning process.

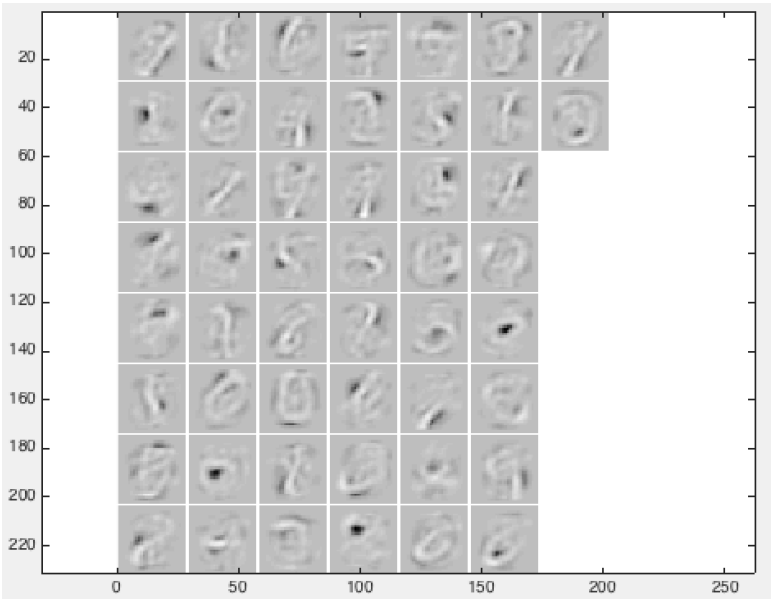


Fig. 3.5 *RBM weights for MNIST recognition*

Below are provided some recognition examples. More spikes represent more concern.

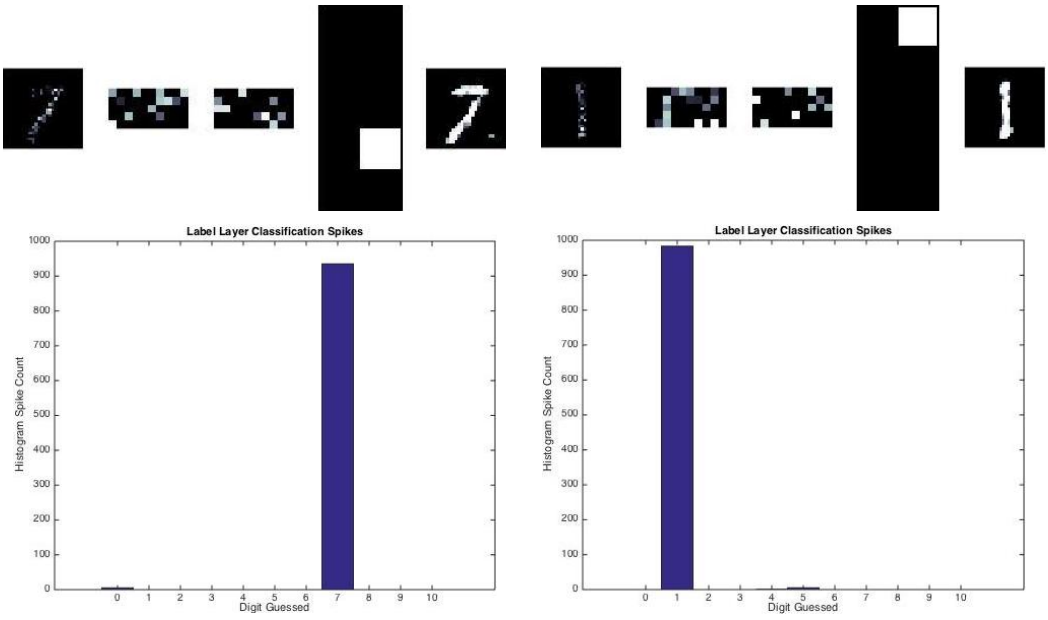


Fig. 3.6 *Recognition of MNIST sample 48 (7)*

Fig. 3.7 *Recognition of MNIST sample 655 (1)*

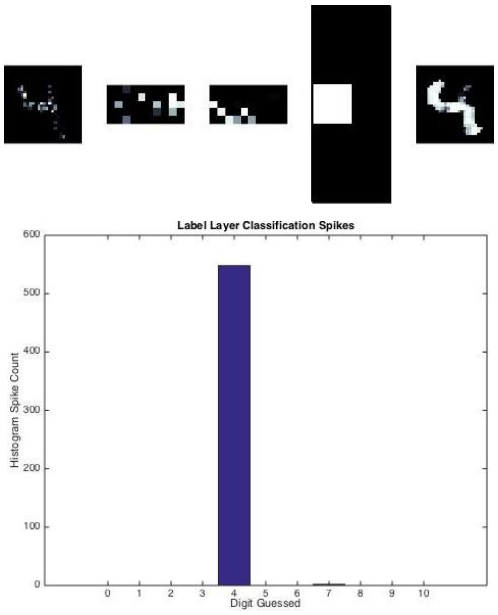


Fig. 3.8 *Recognition of MNIST sample 6592 (4)*

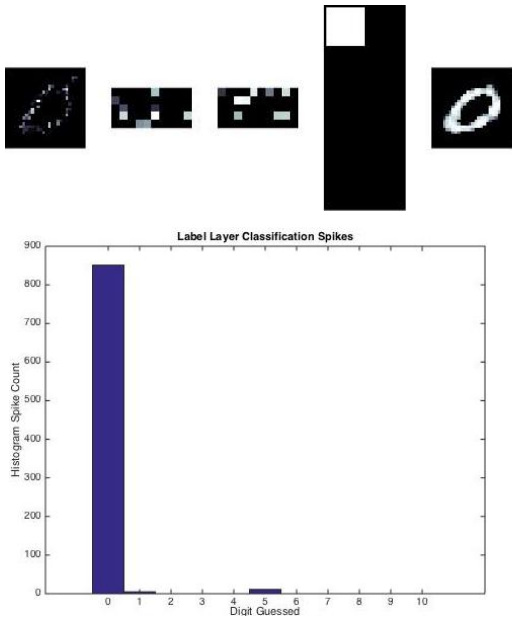


Fig. 3.9 *Recognition of MNIST sample 7362 (0)*

Due to the % of recognition, some numbers cannot be recognized or are recognized wrongly.

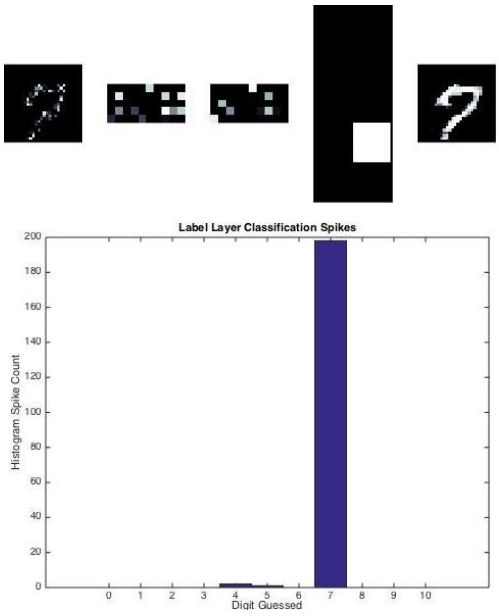


Fig. 3.10 *Recognition of MNIST sample 74 (9)*

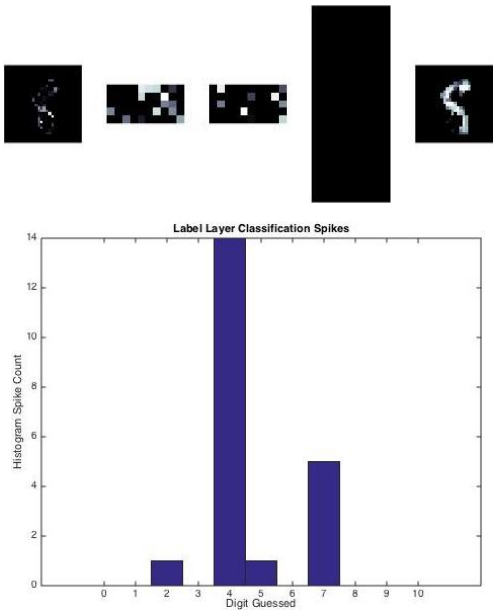


Fig. 3.11 *Recognition of MNIST sample 492 (8)*

This approach allows recognizing MNIST dataset with 90.46% probability, what is a very good percentage. Research team reports that with more advanced setup can be obtained recognition level 95.52% [40].

4. Minimal recognition size

Within discussion about solving capabilities of this approach was asked a question about minimal resolution of input.

4.1 Dataset

The same approach from chapter 6 was used for data generation, but instead of 5x7 input matrix was used 3x3 matrix. This is square form is used for LCD or LED display and named “Fakoo alphabet” [42].

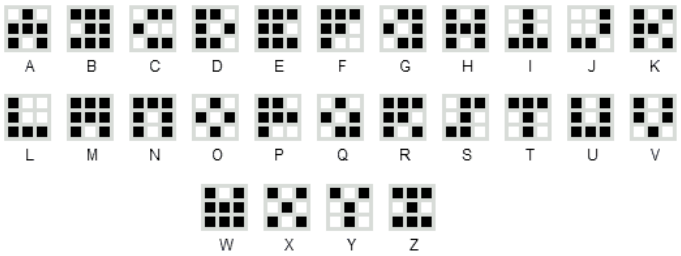


Fig. 4.1 *Fakoo alphabet [42]*

4.2 Setup

Following setup was used for SNN training. Example of program listing can be reviewed in User Guide chapter 5.

```
edbn.sizes = [9 45 52 26];

opts.alpha      = 0.5;
opts.momentum   = 0.5;
opts.f_decay    = 0.006;
opts.f_alpha    = 3;
opts.pcd        = 0.8;
opts.sp         = 0.2;
opts.sp_infl    = 0.7;
opts.ngibbs     = 2;
opts.batchsize  = 13;

opts.numepochs  = 50;
```

Table 4.1 *SNN setup for Fakoo alphabet recognition*

4.3 Result

```
Epoch 40: mean error: 0.03378.
Epoch 41: mean error: 0.03121.
Epoch 42: mean error: 0.05170.
Epoch 43: mean error: 0.04158.
Epoch 44: mean error: 0.03660.
Epoch 45: mean error: 0.04904.
Epoch 46: mean error: 0.04434.
Epoch 47: mean error: 0.04466.
Epoch 48: mean error: 0.04979.
Epoch 49: mean error: 0.04390.
Epoch 50: mean error: 0.04018.
Scored: 100.00
```

Fig. 4.2 *Training result of Fakoo alphabet recognition*

After training was achieved recognition capability: 100%.

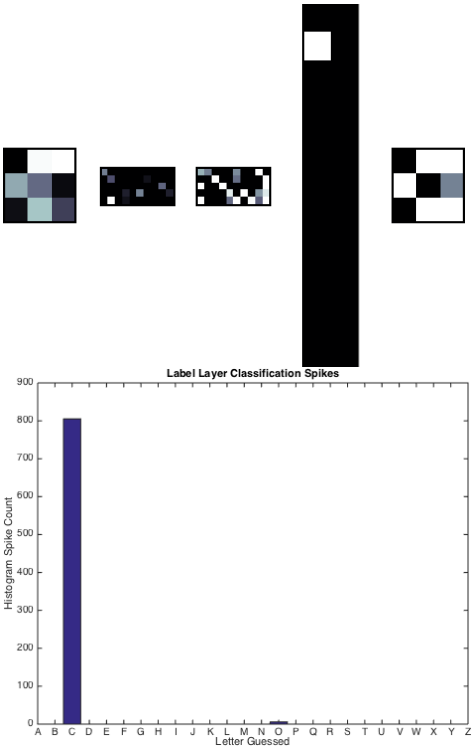


Fig. 4.3 Recognition of Fakoo alphabet sample 3 (C)

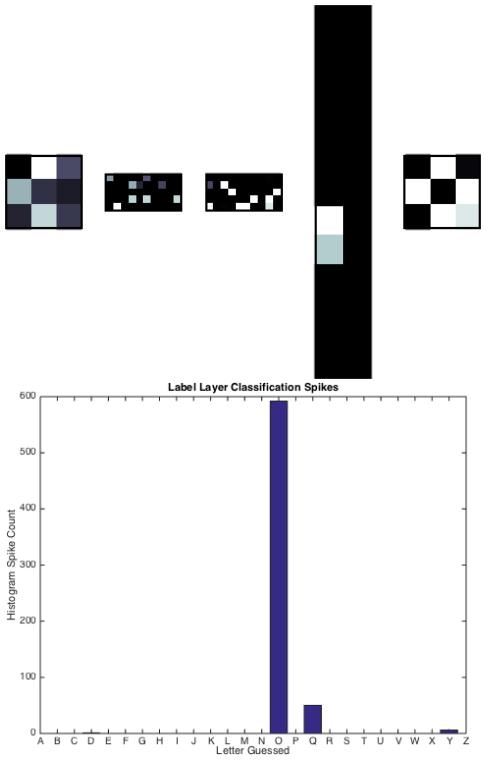


Fig. 4.4 Recognition of Fakoo alphabet sample 15 (O)

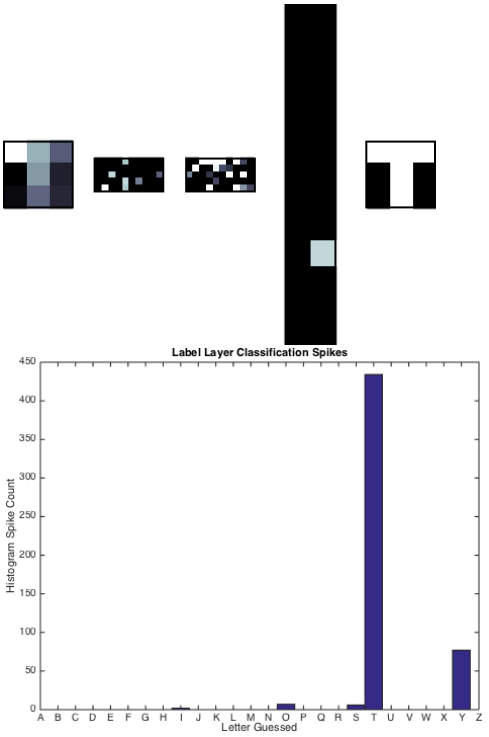


Fig. 4.5 Recognition of Fakoo alphabet sample 20 (T)

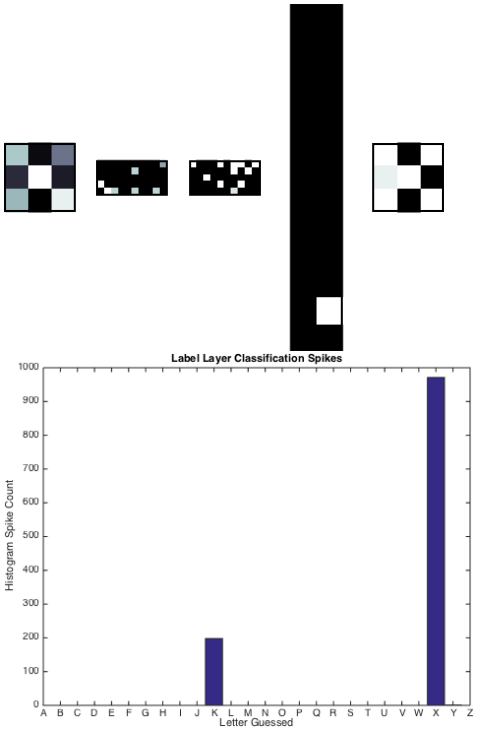


Fig. 4.6 Recognition of Fakoo alphabet sample 24 (X)

Considering very small size of an input image, this is quite impressive that all characters was recognized.

5. Character recognition

The next task was taken from TUT course named Intelligent Control Systems (ISS0023) [43] by Eduard Petlenkov. This course provides overview of artificial intelligence methods based classification and recognition techniques.

5.1 Task

Course task is to train FFNN for alphabet character recognition in two ways (supervised and unsupervised). My aim is to solve the same task with spiking neural network and compare opportunities of this networks by accuracy, used time and number of epoch.

All character is given as 5x7 matrix. In total 26 samples.

```
letterA =  [0 0 1 0 0 ...
            0 1 0 1 0 ...
            0 1 0 1 0 ...
            1 0 0 0 1 ...
            1 1 1 1 1 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ]';

letterB =  [1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 1 ...
            1 1 1 1 0 ]';

letterC =  [0 1 1 1 0 ...
            1 0 0 0 1 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 0 ...
            1 0 0 0 1 ...
            0 1 1 1 0 ]';
```

Fig. 5.1 *Example of characters A, B and C as matrix*

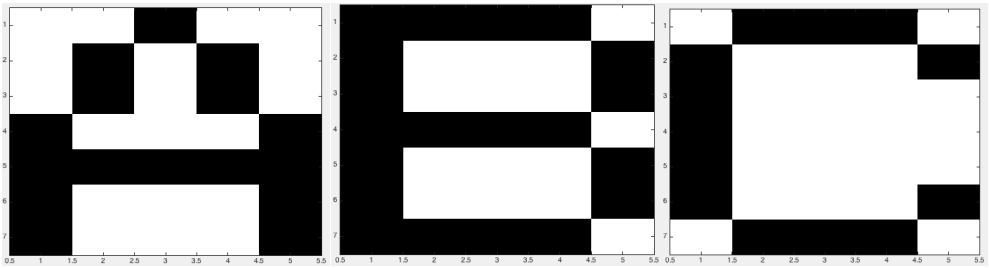


Fig. 5.2 *Example of characters A, B and C as image*

For training purpose, set consist of pure image and images with applied noise 5%,10%,20% and 30%. Testing data are designed with 22% noise.

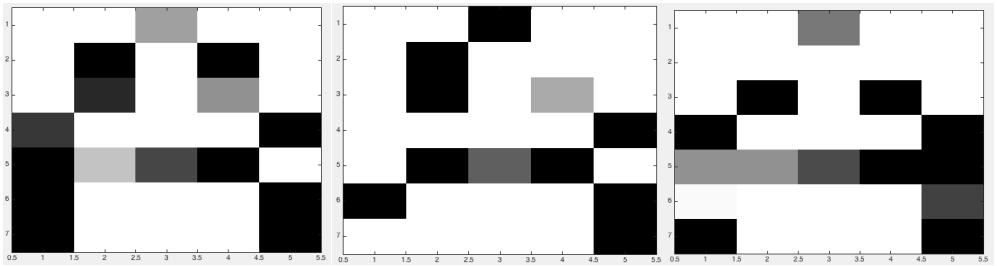


Fig. 5.3 *Letter A with noise 20%, 30% and 22%*

5.2 Training results with FeedForward Neural Network

After training and verification, were achieved following results:

- Accuracy – 100% (all letters recognized correctly)

```
test_result =

  Columns 1 through 13

     1     2     3     4     5     6     7     8     9    10    11    12    13

  Columns 14 through 26

    14    15    16    17    18    19    20    21    22    23    24    25    26
```

Fig. 5.4 *FFNN lab task accuracy result*

- Time – 3.571 seconds (CPU usage)

Function Name	Calls	Total Time
recognition_by_FF_net	1	3.571 s

Fig. 5.5 *FFNN lab task time result*

- Number of epoch – 1744

Epoch:	0	1744 iterations	5000
Time:		0:00:03	
Performance:	0.373	1.93e-12	0.00
Gradient:	0.0781	9.87e-13	1.00e-12
Validation Checks:	0	0	6

Fig. 5.6 *FFNN lab task epoch*

Examples of test data:

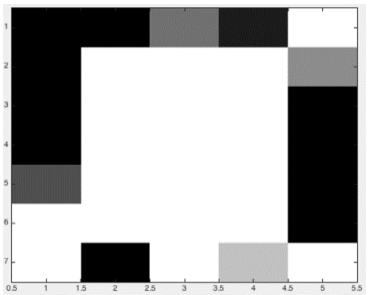


Fig. 5.7 *Lab task testing data sample 4 (D)*

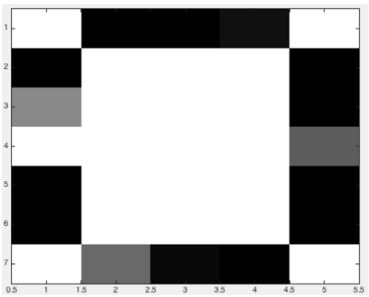


Fig. 5.8 *Lab task testing data sample 15 (O)*

As we can see, Feedforward neural network performed training and recognition task as needed.

5.3 Training with Self-Learning Neural Network

Here will be presented optimal epoch number for self-learning neural network training. To find it, I will use comparison between clean letter and with noise.

for i=1:26
 	
test=alphabet(:,i);
test_n=test+randn(35,1)*0.22;
	Cycle for every letter
	
	Clean letter
	Same latter with 22% noise

t=sim(net_c,test); t_n=sim(net_c,test_n);	Recognition of clear letter Recognition of letter with noise

test_out (i) = vec2ind(t);
test_out_n (i) = vec2ind(t_n);
end
	Writing recognition results

test_out (26) = vec2ind(t) test_out_n (26) = vec2ind(t_n)	Final result of recognition
test_out == test_out_n match=isequal(test_out,test_out_n)	Comparison side-by-side Does 2 vectors equal or not

Table 5.1 *Lab task self-learning neural network recognition verification*

Epoch = 10, not enough for all character recognition.

```
epoch =
    10

test_out =
     5    15    24    21    20    26    24     1    22    23     7    10    18    17    21     4     2    12    13     6    11    25     9    14    16     8

test_out_n =
     5    15     3    21    26    26    19     1    22    23     7    10    18    17    19    26     2    12    13    23    19    25     9    14    16     8

s_b_s =
     1     1     0     1     0     1     0     1     1     1     1     1     1     1     0     0     1     1     1     0     0     1     1     1     1     1

match =
     0
```

Fig. 5.9 *Lab task self-learning verification with 10 epoch*

Epoch=20, also not enough.

```
epoch =
    20

test_out =
     8    15     1    23    12    20    17    25    22    19     6     9    14     2     5     4    11    18    16     3    24    26    21     7    10    13

test_out_n =
     8    15     1     5    12    20    17    25    22    19     6     9    14     2     5     4    11    18    16     3    24    26    21     7    10    13

s_b_s =
     1     1     1     0     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

match =
     0
```

Fig. 5.10 *Lab task self-learning verification with 20 epoch*

Epoch=25, not enough.

```
epoch =
    25

test_out =
    11    17    21    22    25    10    14    19    20    23     1     8    24    15     4    26     2    13     9     7    12     6    18     3    16     5

test_out_n =
    11    17    14    14    25    10    14    19    20    23     1     8    24    14    14    26     2    13     9     7    12     6    18     3    16     5

s_b_s =
     1     1     0     0     1     1     1     1     1     1     1     1     1     0     0     1     1     1     1     1     1     1     1     1     1     1

match =
     0
```

Fig. 5.11 *Lab task self-learning verification with 25 epoch*

Epoch=30, enough, all characters recognized.

```
epoch =
    30

test_out =
    13    17    22    10    21    25     4    20    24    23     5     2    18    14    26     3     6    19    11     7     8    15     1    12     9    16

test_out_n =
    13    17    22    10    21    25     4    20    24    23     5     2    18    14    26     3     6    19    11     7     8    15     1    12     9    16

s_b_s =
     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1     1

match =
     1
```

Fig. 5.12 *Lab task self-learning verification with 30 epoch*

30 epochs were enough for correct recognition of all character and will be used in recognition methods comparison for self-learning network.

After training and verification, were achieved following results:

- Accuracy – 100% (all letters recognized correctly, attempts are presented in Appendix 1)

```
test_out_n =

  Columns 1 through 13

    13    17    22    10    21    25     4    20    24    23     5     2    18

  Columns 14 through 26

    14    26     3     6    19    11     7     8    15     1    12     9    16
```

Fig. 5.13 *SLNN lab task accuracy result*

- Time – 4.590 seconds (CPU usage)

Function Name	Calls	Total Time
recognition_by_SL_net	1	4.590 s

Fig. 5.14 *SLNN lab task time result*

- Number of epoch – 30

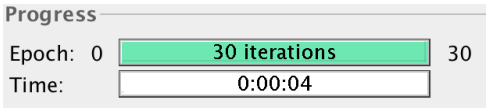


Fig. 5.15 *SLNN lab task epoch*

Examples of test data:

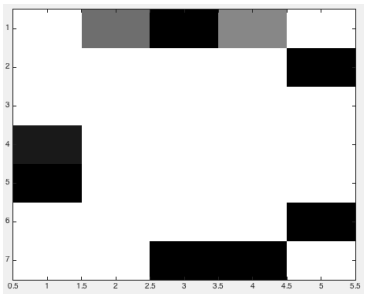


Fig. 5.16 *Lab task testing data sample 3 (C)*

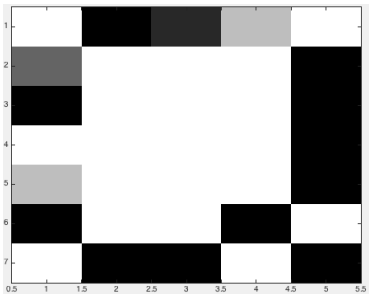


Fig. 5.17 *Lab task testing data sample 17 (Q)*

Self-learning network also performed well.

5.4 Training with Spiking Neural Network

Detailed setup and training process presented in User Guide chapter 7.

After training and verification, were achieved following results:

- Accuracy – 100% (all letters recognized correctly)

```
Scored: 100.00
```

Fig. 5.18 *SNN lab task accuracy result*

- Time – 1.691 seconds (CPU usage)

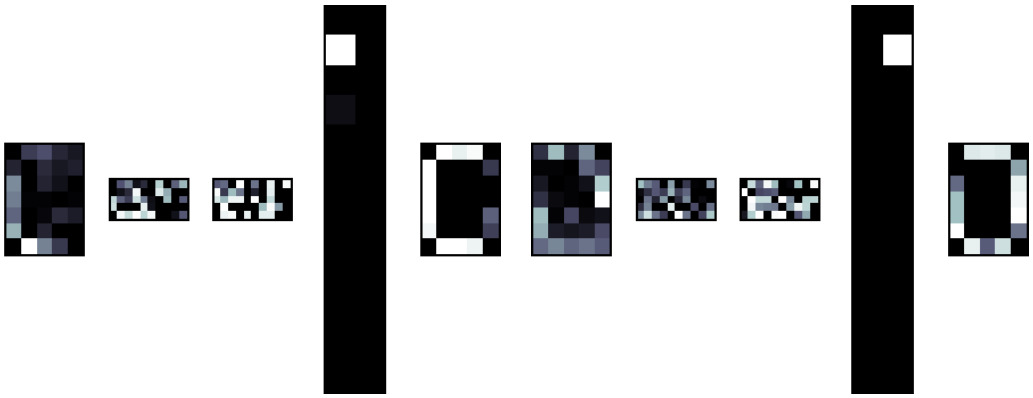
Function Name	Calls	**Total Time**
recognition_by_SNN_net	1	1.691 s

Fig. 5.19 *SNN lab task time result*

- Number of epoch – 50 (25 epochs for each RBM)

```
Epoch 15: mean error: 0.04370.  Epoch 15: mean error: 0.03106.
Epoch 16: mean error: 0.05020.  Epoch 16: mean error: 0.03392.
Epoch 17: mean error: 0.05160.  Epoch 17: mean error: 0.02881.
Epoch 18: mean error: 0.08228.  Epoch 18: mean error: 0.02936.
Epoch 19: mean error: 0.07698.  Epoch 19: mean error: 0.03568.
Epoch 20: mean error: 0.07164.  Epoch 20: mean error: 0.02465.
Epoch 21: mean error: 0.06818.  Epoch 21: mean error: 0.02490.
Epoch 22: mean error: 0.04471.  Epoch 22: mean error: 0.04318.
Epoch 23: mean error: 0.06470.  Epoch 23: mean error: 0.04054.
Epoch 24: mean error: 0.04391.  Epoch 24: mean error: 0.04676.
Epoch 25: mean error: 0.07901.  Epoch 25: mean error: 0.02586.
```

Fig. 5.20 *SNN lab task epoch*



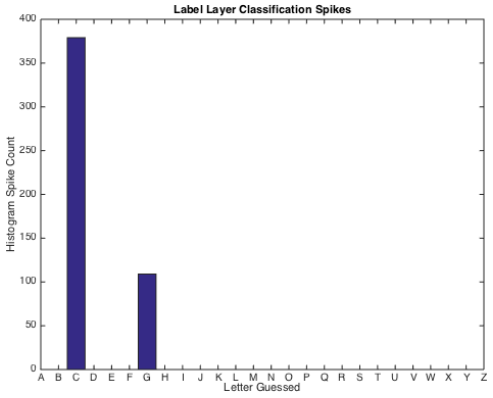


Fig. 5.21 *Recognition of sample 3 (C)*

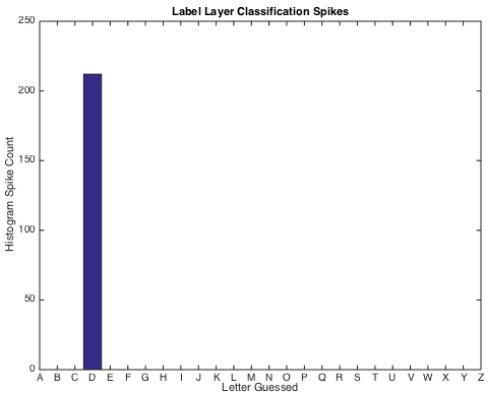
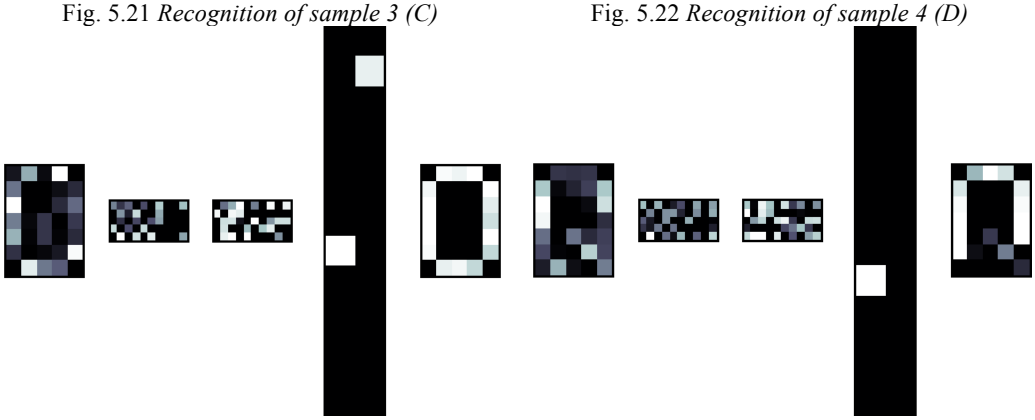


Fig. 5.22 *Recognition of sample 4 (D)*



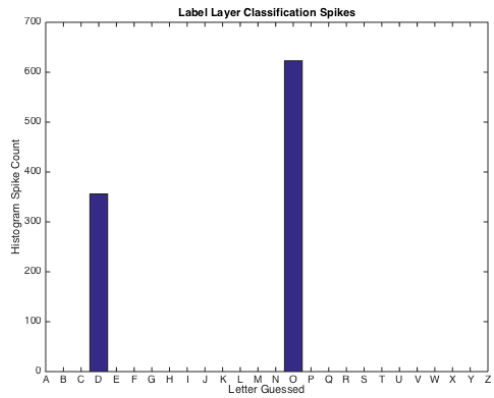


Fig. 5.23 *Recognition of sample 15 (O)*

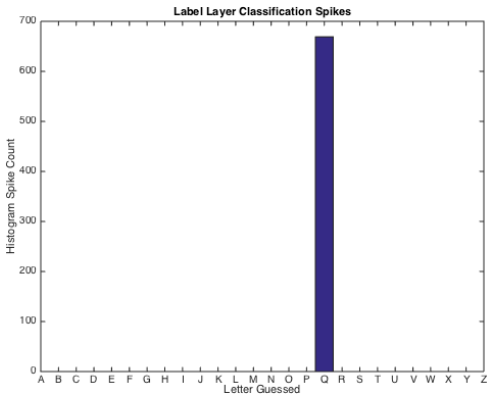


Fig. 5.24 *Recognition of sample 17 (Q)*

5.5 Comparison of the results

Obtained learning results can be easily compared.

	Accuracy	Neurons in hidden layer	Time	Epoch
Feedforward neural network	100%	35+26	3.571	1744
Self-learning neural network	100%	26	4.590	30
Spiking neural network	100%	35+26	1.691	50

Table 5.2 *Lab task result comparison*

Based on the obtained data, spiking neural network can be considered as one of the solution of image recognition problem and can be demonstrated as entry point for more advanced neural network.

6. Summary

In this thesis was presented one of the approaches in recognition technics. Wide range of future development in this field can be implemented in everyday life. But the complexity of the recognition and classification process requires the synergy of different methodology from a different field. At the moment neuromorphic computing faces the problem of big data. Existing applications and solutions give as a place of improvement and development to overcome the barrier and reach a new level in artificial neural networks.

Spiking neural network can be used in digitizing handwriting materials or speech recognition. Ability to make real-time recognition great opportunity to use in mobile applications or in robotics. A lot of existing solution require a data connection with external servers. This connection can be restricted, for example in remote areas.

This particular solution is a proof-of-concept and good starting point for learning spiking neural network recognition opportunities.

As a future development of this real-time solution can be adjusting other external equipment and processing more complex problem.

References

[1] Vinícius Gonçalves Maltarollo, Káthia Maria Honório and Albérico Borges Ferreira da Silva, *"Applications of Artificial Neural Networks in Chemical Problems"*, Artificial Neural Networks - Architectures and Applications, pp. 204-206, 2013

[2] Intel software developer zone, *"Can Technology Replace The Eye?",* https://software.intel.com/en-us/articles/can-technology-replace-the-eye (30.05.2017)

[3] Stanford university class *"The Intellectual Excitement of Computer Science"* https://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/Architecture/feedforward.html (30.05.2017)

[4] *Feedforward neural network in javascript* https://robertbeisicht.wordpress.com/2014/07/04/feed-forward-neural-network-in-javascript/ (30.05.2017)

[5] Mark J.L. Orr, *"Introduction to Radial Basis Function Networks",* Technical report, 1996

[6] *Radial Basis Function Network (RBFN) Tutorial* http://mccormickml.com/2013/08/15/radial-basis-function-network-rbfn-tutorial/ (30.05.2017)

[7] *Recurrent neural networks tutorial, part 1 – introduction to rnns* http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/ (30.05.2017)

[8] *Deep Visual-Semantic Alignments for Generating Image Descriptions* http://cs.stanford.edu/people/karpathy/deepimagesent/ (30.05.2017)

[9] *Physical neural network liquid state machine utilizing nanotechnology* https://www.google.com/patents/US7392230 (30.05.2017)

[10] Marcus Waibel, Dario Floreano and Laurent Keller, *"A Quantitative Test of Hamilton's Rule for the Evolution of Altruism"*, PLoS Biology, 2011

[11] Linda Corucci, Andrea Masini, Marco Cococcioni, *"Approaching bathymetry estimation from high resolution multispectral satellite images using a neuro-fuzzy technique"*, Journal of Applied Remote Sensing Volume 5, Issue 1, 2011

[12] Spiros Ioannou, Amaryllis Raouzaiou, Vasilis Tzouvaras, Theofilos Mailis, Kostas Karpouzis, Stefanos Kollias, *"Emotion recognition through facial expression analysis based on a neurofuzzy network",* Neural Networks. 18 (4), 2005

[13] Jeff Hawkins, "*On Intelligence"*, NY:Times Books, 2004

[14] Zhou Lai, Gu Hongbin and Niu Ben, *"Visual Hand Pose Estimation Based on Hierarchical Temporal Memory in Virtual Reality Cockpit Simulator"*, Information Technology Journal 10, 2011

[15] Wolfgang Maass, *"Networks of spiking neurons: The third generation of neural network models"*, Neural Networks. 10 (9),1997

[16] Chung-Chuan Lo's lab, *Drosophila whole brain simulation,* http://life.nthu.edu.tw/~lablcc/research_ch.html (30.05.2017)

[17] MIT Technology review, *Deep learning* https://www.technologyreview.com/s/513696/deep-learning/ (30.05.2017)

[18] Wulfram Gerstner, Richard Kempter, J. Leo van Hemmen and Hermann Wagner, "*Hebbian Learning of Pulse Timing in the Barn Owl Auditory",* MIT press, 1999

[19] Fred Rieke, David Warland, Rob de Ruyter van Steveninck, William Bialek, *"Spikes: Exploring the Neural Code"*, MIT Press, 1997

[20] David Ferster, Nelson Spruston, "*Cracking the neuronal code"*, Science, vol. 270 p.756- 757, 1995

[21] *Колесницкий О. К., Бокоцей И. В., Яремчук С. С., "Аппаратная реализация элементов импульсных нейронных сетей с использованием биспин-приборов, Часть 1",* XII Всеросиийская научно-техническая конференция «Нейроинформатика», 2010

[22] Julie Dethier, Paul Nuyujukian, Chris Eliasmith, Terrence C. Stewart, Shauki A. Elasaad, Krishna V. Shenoy and Kwabena A. Boahen, *"A Brain-Machine Interface Operating with a Real-Time Spiking Neural Network Control Algorithm",* Advances in neural information processing systems, 2011

[23] Mehmet Kocaturk, Halil Ozcan Gulcur and Resit Canbeyli, *"Toward building hybrid biological/in silico neural networks for motor neuroprosthetic control*", Frontiers in Neurorobotics 9, 2015

[24] BrainCorp https://www.braincorp.com/technology/ (30.05.2017)

[25] Electronics360, *Intel Follows Qualcomm Down Neural Network Path* http://electronics360.globalspec.com/article/4318/intel-follows-qualcomm-down-neural-network-path (30.05.2017)

[26] *Introducing Qualcomm Zeroth Processors: Brain-Inspired Computing* https://www.qualcomm.com/news/onq/2013/10/10/introducing-qualcomm-zeroth-processors-brain-inspired-computing (30.05.2017)

[27] Taras Iakymchuk, Alfredo Rosado-Muñoz, Juan F. Guerrero-Martínez, *"Simplified spiking neural network architecture and STDP learning algorithm applied to image classification"*, EURASIP Journal on Image and Video Processing, 2015

[28] IEEE spectrum, *How IBM Got Brainlike Efficiency From the TrueNorth Chip* http://spectrum.ieee.org/computing/hardware/how-ibm-got-brainlike-efficiency-from-the-truenorth-chip (30.05.2017)

[29] Lawrence Livermore National Laboratory, *"The 16-chip IBM TrueNorth platform"* https://www.llnl.gov/news/lawrence-livermore-and-ibm-collaborate-build-new-brain-inspired-supercomputer (30.05.2017)

[30] IBM Research, *Neurosynaptic systems*, http://www.research.ibm.com/cognitive-computing/neurosynaptic-chips.shtml#fbid=w89hb0votSW (30.05.2017)

[31] Peter O'Connor, *"Real-Time Classification and Sensor Fusion with a Spiking Deep Belief Network"* Frontiers in Neuroscience 7, 2013

[32] Yosua Bengio, Pascal Lamblin, Dan Popovici and Hugo Larochelle, *"Greedy layer-wise training of deep networks"*, Advances in Neural Information Processing Systems 19, MIT Press, 2006

[33] Vinod Nair, Geoffrey E. Hinton, *"Rectified linear units improve Restricted Boltzmann Machines"*, Proceedings of ICML, 807–814, 2010

[34] Geoffrey E. Hinton, Terrence J. Sejnowski, *"Learning and Relearning in Boltzmann Machines"*,MIT Press, pp. 282–317, 1986

[35]Geoffrey E. Hinton, Ruslan R. Salakhutdinov, *"Reducing the dimensionality of data with neural networks"*, Science 313, pp. 504–507, 2006

[36]Geoffrey E. Hinton, Simon Osindero, Yee Whey Teh, *"A fast learning algorithm for deep belief nets"*, Neural Comput. 18, 1527–1554, 2006

[37] Wulfram Gerstner, Werner Kistler, *"Spiking Neuron Models. Single Neurons, Populations, Plasticity"*, Cambridge University Press, 2002

[38] Arnold J. F. Siegert, *"On the first passage time probability problem"*, Phys, 1951

[39] Florian Jug, Matthew Cook and Angelika Steger, *"Recurrent competitive networks can learn locally excitatory topologies"*, International Joint Conference on Neural Networks, 2012

[40] SNN model for MATLAB https://github.com/dannyneil/edbn (30.05.2017)

[41] Deep Learning Toolbox for MATLAB https://se.mathworks.com/MATLABcentral/fileexchange/38310-deep-learning-toolbox (30.05.2017)

[42] Fakoo alphabet example http://www.fakoo.de/fakoo/fakoo-alphabet_en.html (30.05.2017)

[43] Intelligent Control Systems (ISS0023) course homepage http://a-lab.ee/edu/ISS0023 (30.05.2017)

[44] Spiking neural network files https://github.com/dannyneil/edbn/archive/master.zip (30.05.2017)

[45] Spiking neural network solution introduction https://github.com/dannyneil/edbn/blob/master/README.md (30.05.2017)

User guide

Following manual is designed as entry point for those, who wants to try and understand basics of this particular solution for image recognition using spiking neural network in MATLAB environment.

1. Software

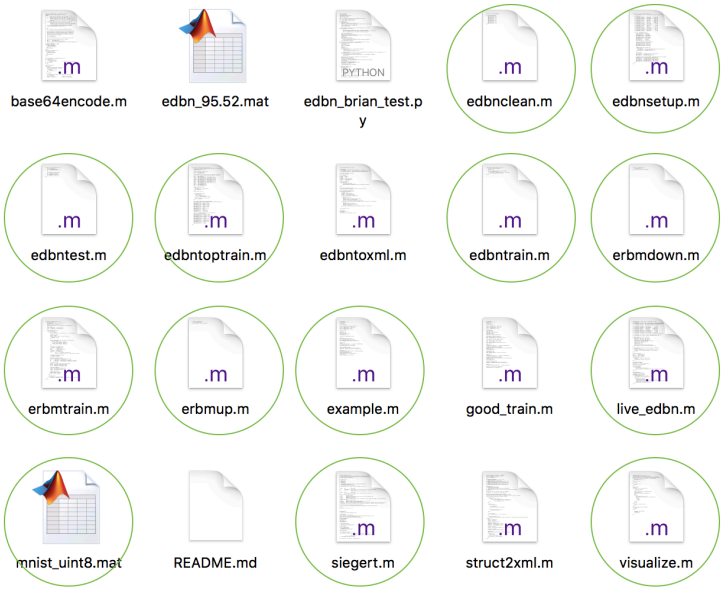
First thing that you must have is MATLAB/SIMULINK program. I have used version R2014b. Also you must have following toolboxes:
To check which modules are installed, enter in Command Window "ver"

```
>> ver
-----------------------------------------------------------------------------------------------
MATLAB Version: 8.4.0.150421 (R2014b)
Operating System: Mac OS X  Version: 10.12.6 Build: 16G8c
Java Version: Java 1.7.0_55-b13 with Oracle Corporation Java HotSpot(TM) 64-Bit Server VM mixed mode
-----------------------------------------------------------------------------------------------
MATLAB                                                Version 8.4        (R2014b)
Simulink                                              Version 8.4        (R2014b)
Statistics Toolbox                                    Version 9.1        (R2014b)
Symbolic Math Toolbox                                 Version 6.1        (R2014b)
```

2. Package

Since at the moment there is no proper toolbox for spiking neural network in MATLAB, main files can be downloaded from Github repository [44].

After download and extraction, you should have folder with name "edbn-master" with following set of files.

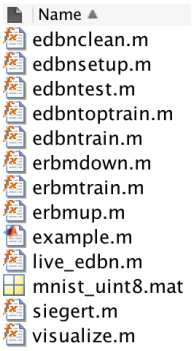


Optional: because this solution is a part of bigger project, only files with green cycle is needed for work

3. MATLAB

After extraction “edbn-master” folder must be placed as a current working folder in MATLAB. Easiest way to do it is drag-n-drop folder in “Current Folder” section in MATLAB. After that, double tap on this folder in MATLAB.

If all previous steps done correctly, in MATLAB should be followed file structure.



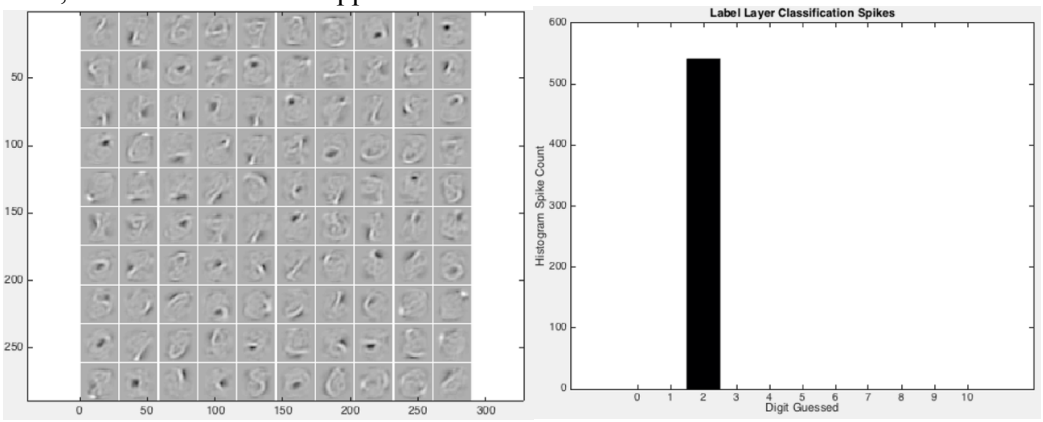
3.1 Installation verification

To proof, if everything is set correctly, enter “example” in Command Window and press enter.

Training of the network takes time. At the end in the command window you should see similar result, but mean error, score and time can be different.

```
>> example
Beginning training.
Epoch 1: mean error: 0.00843.
Epoch 2: mean error: 0.00862.
Epoch 3: mean error: 0.00716.
Epoch 4: mean error: 0.00628.
Epoch 5: mean error: 0.00909.
Epoch 6: mean error: 0.00766.
Scored: 91.80
Completed 2000 input spikes occurring over 4.00 seconds, in 5.402 seconds of real time.
```

Also, 2 new window must appear.



Meaning of this values and graphs will be explained later.

4. Files

Here are listed all used filed with short explanation what this files used for [45].

- **edbnclean.m** - cleans out all the temporary activations to save a minimum-size EDBN file.
- **edbnsetup.m** - initializes the network and load defaults.
- **edbntest.m** – perform testing comparison
- **edbntoptrain.m** - performs supervised training by concatenating the top layer to the top-2 layer, and jointly training a (top-2, top) <-> (top-1) RBM, then unrolling again.
- **edbntrain.m** - performs unsupervised training of the network.
- **erbmup.m / erbmdown.m** - propagates rate-based activations up or down through LIF neurons.
- **erbmtrain.m** - trains a single RBM layer in the DBN. This is the *core* source file for the algorithm.
- **example.m** - runs an example.
- **live_edbn.m** - run the weights on an actual spiking network of neurons.
- **mnist_uint8.mat** – prepared MNIST dataset
- **siegert.m** - calculates the output spike rate of an input rate and input weights for LIF neurons.
- **visualize.m** – visualize weights of the RBM

5. Settings

As was mentioned before, we deal with a part of bigger project. So, to simplify understanding and remove unnecessary errors, some changes in example.m can be made.
Removed code lanes will be highlighted.

5.1 Basic

Main work file in this approach is example.m. Below are explained what is inside this file and how it can be used.

`%% Load paths` `addpath(genpath('.'));`	Returns path to MATLAB toolbox folder and ad this path to search path for this session
`%% Load data` `load mnist_uint8;`	Load MNIST dataset as training and testing data

`% Convert data and rescale between 0 and 0.2`
`train_x = double(train_x) / 255 * 0.2;`
`test_x = double(test_x) / 255 * 0.2;`
`train_y = double(train_y) * 0.2;`
`test_y = double(test_y) * 0.2;`
	Data in dataset in uint8 format and must be converted to double. Also inputs (*_x files) are given from 0 (white) to 255 (black) and must be rescaled for range from 0 to 1. Multiplication by 0.2 is needed for maximizing spike firing.

`%% Train network`
`% Setup`
`rand('seed', 42);`
`clear edbn opts;`
`edbn.sizes = [784 100 10];`
`opts.numepochs = 6;`
	Seeds the random number generator using the nonnegative integer.
	Removes previously entered network settings.
	Set number of inputs, neurons in hidden layer and number of outputs.
	Number of training cycles for each hidden layer.

`[edbn, opts] = edbnsetup(edbn, opts);`	Load setting in network initializing file.

`% Train`
`fprintf('Beginning training.\n');`
`edbn = edbntrain(edbn, train_x, opts);`
`% Use supervised training on the top layer`
`edbn = edbntoptrain(edbn, train_x, opts, train_y);`
	Prints message about training start.
	Perform supervised training of every single layer.
	Train the top layer by merging the top layer to a lower layer and jointly training the set.

`% Show results`
`figure;`
`visualize(edbn.erbm{1}.W');  % Visualize the RBM weights`
`er = edbntest (edbn, test_x, test_y);`
`fprintf('Scored: %2.2f\n', (1-er)*100);`
	Create a figure.
	Shows vector of weigh for each neuron.
	Calculates an error in recognition set.
	Prints message with percent of recognition.

`%% Show the EDBN in action`
`spike_list = live_edbn(edbn, test_x(1, :), opts);`
`output_idxs = (spike_list.layers == numel(edbn.sizes));`
	Show feed with spike activity.

`figure(2); clf;`
`hist(spike_list.addrs(output_idxs) - 1, 0:edbn.sizes(end));`
`xlabel('Digit Guessed');`
`ylabel('Histogram Spike Count');`
`title('Label Layer Classification Spikes');`
	Show result of recognition.

`%% Export to xml` `edbntoxml(edbn, opts, 'mnist_edbn');`	Creates a base64-encoded representation of the network.

5.2 Advanced

More specific setting can be found in edbnsetup.m.

`opts.alpha = 1`	Learning rate
`opts.decay = 0.0001`	Spike decay speed
`opts.momentum = 0.0`	Impuse
`opts.temp = 0.005`	Noise for Siegert function
`opts.tau_m = 5.0`	Membrane time constent
`opts.tau_s = 0.001`	Synaptic response time constant
`opts.t_ref = 0.002`	Absolute refractory time
`opts.v_thr = 0.005`	Threshold of Siegert function
`opts.f_infl = 1`	Fast weight coefficient
`opts.f_decay = 0.05`	Fast weight incorporate decay
`opts.f_alpha = 5`	Fast weight learning rate
`opts.pcd = 1`	Persistent contrastive divergence
`opts.sp = 0.1`	Sparsify
`opts.sp_infl = 0.2`	Sparsify fast weight coefficient
`opts.ngibbs = 2`	Fast weight restriction for obtaining model sample
`opts.initscl = 0.01`	Weight coefficient
`opts.batchsize = 50`	Number of training samples in one neuron
`opts.reup = 1`	Train the composite layer
`opts.wtreset = 1`	Weights and biases update

To change recognition feed setting, open live_edbn.m file.

`opts.recreate = 1`	Show recognized image
`opts.timespan = 4`	Time of live feed
`opts.numspikes = 2000`	Number of spikes used
`opts.delay = 0.001`	Delay between spikes firing
`opts.show_dt = 0.010`	Dependency for spike exp(-opts.show_dt / opts.vis_tau);
`opts.vis_tau = 0.05`	
`opts.makespikes = 1`	Create spike proportional to intensity
`opts.makevisdim = 1`	Build show dimensions

6 Results

6.1 Training

After execution “example.m” in Command Window will appear:

Beginning training.
Training has started.

Epoch 1: mean error: 0.00843.
Passed training cycles and error between desired and obtained error. Lower value means better result.

6.2 Feedback

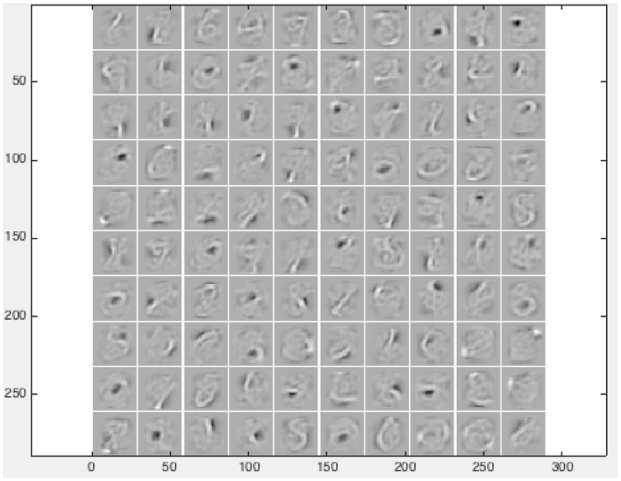
When training is finished, following information will be available in Command Window.

Scored: 91.80
Percentage of recognized inputs.

Completed 2000 input spikes occurring over 4.00 seconds, in 5.557 seconds of real time.
Number of spikes and time, used for active processing.

6.3 Visual representation

First graph – weight coefficients

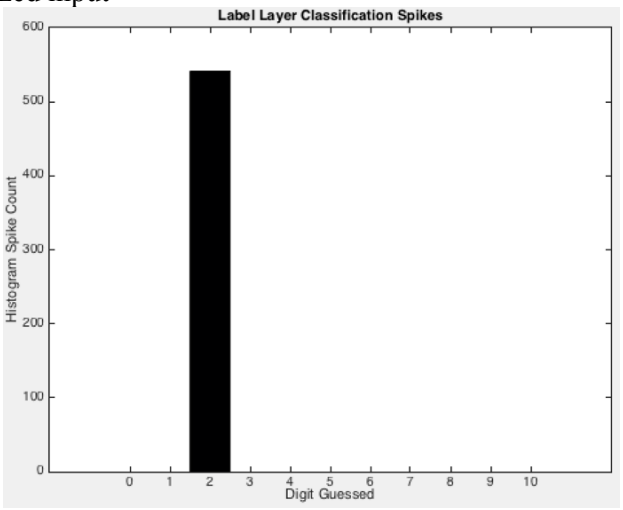


Each square represents vector of weight merged into each neuron.

Second graph – spike activity

Input image processing with spikes	Inner spike activity	Recognized input	Restored input

Third graph – recognized input



Number of activated spikes used for recognition.

7 Laboratory work

As a part of TUT course named Intelligent Control Systems (ISS0023) below will be provided example of alphabet recognition.

7.1 Task

The task is to train spiking neural network for character recognition. Every character is given as 5x7 matrix. In total 26 samples.

7.2 Training and testing data

For dataset creation will be used lectures materials ([43]- materials - laboratory works – image recognition lab). From the archive we will use letter.m (dataset) and recognition_by_FF_net (training and test data).

Letter.m file must be placed in the same folder where are files from Section 3 of this guide.

After merging letter.m should be look like:

`letterA = [0 0 1 0 0 ...`
`0 1 0 1 0 ...`
`0 1 0 1 0 ...`
`1 0 0 0 1 ...`
`1 1 1 1 1 ...`
`1 0 0 0 1 ...`
`1 0 0 0 1 ]';`
	Letter A as 5x7 matrix written as vector

…	Letters from A to Z

`letterZ = [1 1 1 1 1 ...`
`0 0 0 0 1 ...`
`0 0 0 1 0 ...`
`0 0 1 0 0 ...`
`0 1 0 0 0 ...`
`1 0 0 0 0 ...`
`1 1 1 1 1 ]';`
	Letter Z as 5x7 matrix written as vector

`alphabet =`
`[letterA,letterB,letterC,letterD,`
`letterE,letterF,letterG,letterH,...`
`letterI,letterJ,letterK,letterL,`
`letterM,letterN,letterO,letterP,...`
`letterQ,letterR,letterS,letterT,`
`letterU,letterV,letterW,letterX,...`
`letterY,letterZ];`
	Dataset

`targets = eye(26);`	Matrix with 1 on diagonal

`P=[alphabet, alphabet+randn(35,26)*0.05,...`
`alphabet+randn(35,26)*0.1,...`
`alphabet+randn(35,26)*0.2,...`
`alphabet+randn(35,26)*0.3,...`
`];`
	Training data generation

`T=[targets targets targets targets targets ];`	Training data answers
`test_data=alphabet+randn(35,26)*0.22;`	Test data generation
`test=eye(26);`	Test data answers, for error calculation

7.3 Setup

To run Spiking neural network with this dataset changes in example.m must be applied. Main modifiers will be highlited.

`%% Load data` `run letters.m;`	Load alphabet dataset as training and testing data

`%% Convert data  %without advanced better use 0.3`
`train_x = double(abs(P.'))*0.4;`
`test_x  = double(abs(test_data.'))*0.4;`
`train_y = double(abs(T.'))*0.4;`
`test_y  = double(abs(test.'))*0.4;`
	(*.') is needed for input, because system accept sample as row. In dataset samples stored as column. abs(*) removes negative values.
	Multiplication by 0.4 is needed for maximizing spike firing.

`%% Train network`
`% Setup`
`rand('seed', 42);`
`clear edbn opts;`
`edbn.sizes = [35 35 26 26];%without advanced better use 35 50 50 26`
`opts.alpha  = 0.4;`
`opts.momentum  = 0.3;`
`opts.f_decay  = 0.003;`
`opts.f_alpha  = 1;`
`opts.pcd  = 0.6;`
`opts.sp  = 0.1;`
`opts.sp_infl  = 0.9;`
`opts.ngibbs  = 3;`
`opts.batchsize = 13; % Can be 1,2,5,10,13,26,65,120`
 	
`opts.numepochs = 25; %without advanced better use 50`
	Seeds the random number generator using the nonnegative integer.
	Removes previously entered network settings.
	Set number of inputs, neurons in hidden layer and number of outputs.
	Number of training cycles for each hidden layer.
	Number of samples per neuron.

`[edbn, opts] = edbnsetup(edbn, opts);`	Load setting in network initializing file.

`% Train`
`fprintf('Beginning training.\n');`
`edbn = edbntrain(edbn, train_x, opts);`
`% Use supervised training on the top layer`
`edbn = edbntoptrain(edbn, train_x, opts, train_y);`
	Prints message about training start.
	Perform supervised training of every single layer.
	Train the top layer by merging the top layer to a lower layer and jointly training the set.

`% Show results`
`figure;`
`visualize(edbn.erbm{1}.W');  %  Visualize the RBM weights`
`er = edbntest (edbn, test_x, test_y);`
`fprintf('Scored: %2.2f\n', (1-er)*100);`
	Create a figure.
	Shows vector of weigh for each neuron.
	Calculates an error in recognition set.
	Prints message with percent of recognition.

`%% Show the EDBN in action`
`spike_list = live_edbn(edbn, test_x(1, :), opts);`
`output_idxs = (spike_list.layers == numel(edbn.sizes));`
	Show feed with spike activity.
	
	Highlighted digit represents testing sample.
	In this case can be changed from 1 up to 26.

`figure(2);`
`hist(spike_list.addrs(output_idxs) - 1, 0:edbn.sizes(end));`
`xlabel('Letter Guessed');`
`ylabel('Histogram Spike Count');`
`xlim([0 25]);`
`set (gca,'xtick', [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25]);`
`set (gca,'xtickLabel',{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'});`
`title('Label Layer Classification Spikes');`
	Show result of recognition.
	
	Add scale tick for each alphabet letter.
	
	Name this tick with letter

For more deep setup, see chapter 5.2 of this guide.

To achieve better visualization, in live_edbn.m file in section «% Build show dimensions» is needed to change

```
opts.show_dims{i} = [prod(factors(2:2:end)) prod(factors(1:2:end))];
```

to

```
opts.show_dims{i} = [prod(factors(1:2:end)) prod(factors(2:2:end))];
```

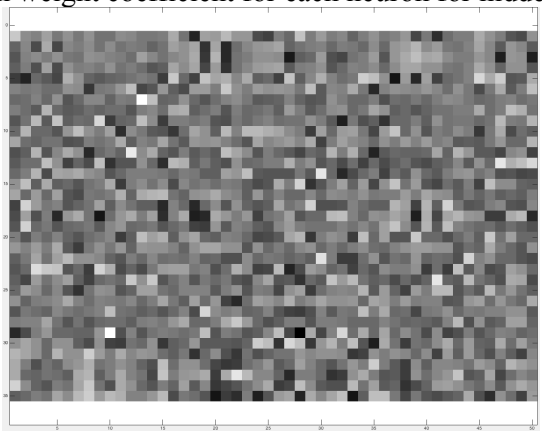
otherwise, matrix will be displayed in first graph as 7x5, but not 5x7.

7.4 Result

After the setup preparation and execution of the modified letter.m will start training process.
Since we used 2 hidden layers and 120 epoch training will take some time.

When training is finished. In Command Window "Score" should be 100.00. It means that all of the 26 testing samples are recognized as correct letter.
Also graphs with recognition results will be created.

First one is graph with weight coefficient for each neuron for hidden layer.



The second group represents recognition process and obtained result. In our case letter A.