
**Faculty of Engineering**

**Department of Computer Engineering**

**M.Sc. Thesis**

**Title of the Thesis**

# Spiking Deep Belief Network

**Supervisor:**

**Dr. Mahmood Ahmadi**

**Advisors:**

**Dr. Arash Ahmadi**

**Professor Philippe Devienne**

**Dr. Mahyar Shahsavari**

**By:**

**Mazdak Fatahi**

**July 2016**

**Abstract:**

Deep learning as a set of algorithms in Machine Learning, has produced state-of-the-art results and their efficiency in various type of applications has been demonstrated. In a biological neural network, both of memory and computational elements are integrated in the body of each neuron. Because of the physical distance between the memory elements and the arithmetic modules in Von-Neumann architecture, implementing deep architectures in general-purpose hardware, despite its accuracy, ignores this biological aspect of neurons. Therefore implementing deep learning algorithms in a specific-purpose hardware with biological inspired neurons not only can utilizes the robustness of deep learning models, but also because of using local analog computation can demonstrate speedup and power efficiency advantages relative to digital systems.

Memristor as a new nano-device provides big opportunity for implementing low power and dense circuits. Implementing biological inspired model of neurons and synaptic connections using Memristor, has proven the efficiency of Crossbars of Memristors in hardware implementation of Spiking Neural Networks.

In this dissertation considering the characteristics of Leaky Integrated-and-Fire (LIF) model as the most popular model of spiking neurons, we have tried to find a method for utilizing Contrastive Divergence (CD) in a network of LIF neurons. There are some problems with using Machine Learning algorithms in spiking model of neural networks. The most challenge with implementing Machine Learning methods using biological inspired model of neurons, is because of their different approaches for updating the synaptic weights. STDP is the most popular method for Spiking Neural Networks (SNN) which uses the spike timing aspect of neural coding, but in this work we provide a framework to use the proposed rate based version of Contrastive Divergence updating weight rule. Respecting to the rate aspect of neural coding, we developed a Spiking Restricted Boltzmann Machine (Spiking RBM) and stacking several RBMs we proposed a Spike-Based Deep Belief Network (S-DBN) with Leaky Integrate-and-Fire neurons.

The proposed framework was evaluated in handwritten digit (MNIST) recognition application task (94.9%). Spike-Based Deep Belief Network (S-DBN), has smoothed the way for utilizing Memristor in a specific hardware implementation of a Deep Spiking Neural Network and has provided a suitable framework to utilizing in deep architectures in a desired Neuromorphic Hardware Accelerator.

## Table of Content

**Content** **page**

## List of Figure

**Content** **page**

## Table of tables

**Content** **page**

# Chapter 1

## Introduction

# 1 Introduction

*"Dose it possible to build machines with human-like intelligence?*" is the basis of *Artificial Intelligence*. This branch of science talks about the methods for using computer systems as intelligent systems. For many years, modelling all the world using computers to have an intelligent system, as the issue of investigations was done by scientists [1]. To achieve this level of perception, storing huge amount of data is needed. To dominate this huge amount of data and doing inference, some learning algorithms were invented. During the last years many efforts were made to improve the learning algorithms, but the challenge is still open [2]. Indeed until now there is no complete learning algorithm to identify the emotions, discovering the semantic concept of images, inferring and predicting. Often exploring such concepts will be done with multi-level of abstraction by human. Understanding the subject of an image in an intelligent system, begins with reading the raw bits of pixels that consisting the image. At this level no specific knowledge will been driven form pixels, but using some feature extraction algorithms we can see some low level geometric feature (ex. Edge detection, orientation detection). In next level applying middle level algorithms help to have some level of abstractions. Using this middle level of abstractions human can explore the concept of an image.

Prerequisite for such an interpretation for an intelligent system is having multi-level architecture that each level of extracting the features reach the system to a higher level of abstraction. The focus of the *deep learning architectures*, is on an autonomous investigating abstraction levels from lowest level to top level, using less human interference in the abstraction levels. The *deep learning approaches*, are looking for learning the high level of abstractions have been built using low level of abstractions. This automatic learning approach for learning the features from various level of abstractions, permits the system to implement complex functions between input and output of a system. Therefore it seems to have an intelligent system, using deep and multi-level architecture is useful.

## 1.1 Artificial intelligence and deep learning

Too many of phenomena in real world are not explainable through exact mathematical equations, but the nature have very perfect answers for them. *Computational Intelligence* is a bunch of nature inspired tools to deal with complex problems. The most important methods for computational intelligence are divided into three classes: *Fuzzy Logics*, *evolutionary computations* and *Artificial Neural Networks*. In this study we concentrate on the approaches

that use Artificial Neural Networks (ANN) to reach Artificial Intelligence (AI). ANN as a kind of computational intelligence, is trying to emulate nearest structure to neural systems of beings (specially the brain structure of human beings), to solve the problems. ANN is consisted of mathematical demonstration of data processing in biological systems [3] and involves a lot of algorithms and architectures. Considering all these aspects and the advancements in neuroscience on one hand and improvement in nanotechnology on the other hand, prepared a new field of engineering called *Neuromorphic* [4]. *Neuromorphic engineering* is affected by sciences such as biology, physics, mathematics and computer. The *Neuromorphic* term was proposed by *Carver Mead* [5] which explore using VLIS to implement neural systems.

The final goal of the investigations is to achieve a structure for imitating the brain which consists of some big projects such as *Human Brian Project (HBP) [6]*,*SpiNNaker[7]* and *Systems of Neuromorphic Adaptive Plastic Scalable Electronics (SyNAPSE) [8]*.

Too many kind of algorithms and architectures was proposed to using neuron like units as computational cells in body of the projects. Generally, in this types of projects there is layer consists of this computational units as the *input layer*. Also there is some other layers as the middle layers which called *hidden layer*, and finally there is layer called *output layer* to show the calculated result. *Feed Forward Network* is a very popular type for this architecture (Figure 1-1). The most famous algorithm for performing calculations in this architecture is *Backpropagation*. Many of ANN implementations only used one hidden layer. These models are *Shallow architectures*. As the architecture gets deeper, it becomes more difficult to obtain good generalization using a Deep NN and this is the main obstacle which prevents using more hidden layers [9]. In [1] Pr. Bengio shows shallow ANN are incapable in handling complex AI problems – also mentioned this is not a reason for putting this architecture aside, and shows using a shallow ANN in the upper level of abstraction which is pre-trained using a hierarchical model, have good results.

This incapability is more obvious in confronting with large amount of data. Using new solutions for storing and integrating data of various fields make large amount of data for processing. Commonly data will be processed to learn features or for data classification. Contrasting with a large amount of data can causes problems in inferring process based on the data, which it can causes more complexity in searching and data classification.

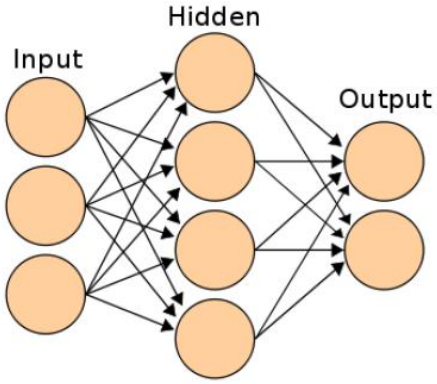


**Figure 1-1: *A* Feed Forward Network (FFN) *with one hidden layer***

As was mentioned in [1, 2] and from biological aspects –such as brain processing which is a layer wised process- and also because of incapability of shallow structures in contrasting with complex and huge problems, in the road-map for achieving AI, using *deep architectures* is inevitable. One of the main problem with using deep structures in last years, was for the absence of a good and perfect algorithm to overcome *deep learning*. Since 2006 -which some algorithms have been proposed [2, 10]- using deep architectures have been more common. One of the best example for using deep architecture is *Google* project called *Google Brain [11]* which is used to mimic some aspects of human brain activity. Google has invested a lot of money attracting professionals such as Pr. Geoffrey Hinton from Toronto University and Pr. Andrew Ng.

Basically, *deep learning* or *hierarchical learning*, indicates type of machine learning which uses multi-layer architecture and can demonstrates high levels of abstraction. Also, in many applications we have a lot of unlabeled data and using *supervised learning algorithms* is impossible. In such cases using some structures which can generate a good representation of system and can extract powerful features from data, is needed. As was mentioned in [2] theoretical results show that using deep architectures which is consist of some layers of nonlinear operators, can generate high level of abstraction. The deepness can be reached repeating the hidden layers. Increasing the number of layers in ANN proposed new structure called *deep networks.* The proposed model can increase the machine learning capability and provide *deep learning.* This learning method was so important which was introduced as one of the ten breakthrough technologies of 2013 by MIT [12].

The goal of deep learning that was introduced as a new filed in machine learning, is to be more close to main goal: *Artificial Intelligence.*

### 1.1.1 Problem Statement

Since in the deep neural networks the philosophy of moving toward AI was kept, and also inspired by biological structure of brain, we are trying to make closer this deep structure to biological model using biological models of neurons called *Spiking Neurons*.

*Spiking Neural Networks (SNN)* in contrast to Artificial Neural Networks (ANN), is more close to biological neurons. In SNN the neurons communicate with each other using very small electrical pulses called *Spike* or *Action Potential*. Generally researcher proposed some circuits to imitate how the real neurons can learn. There are some perfect models for spiking neurons which we will discussed in next the following chapters. The important aspect of spiking models is the ability for hardware implementation. ANN implementation is possible using traditional computers, GPU or FPGA, but these kind of implementations use a multi-purpose hardware for a specific application which is not as efficient as using a specific hardware for neural networks applications. Since using Neuromorphic SNN can be implemented in VLSI, it seem it's possible to have a specific *Hardware Accelerator* called *Neuromorphic Accelerator* neural network applications.

With respect to this possibility for Spiking Neural Networks, and also regarding the efficiency of Deep Learning algorithms and architectures, in this thesis we are looking for a way to use both of them in a project as a Deep Hardware Accelerator, but there are some challenges which two of them are more important :

- In Artificial Neural Networks (ANN) and Machine Learning (ML) domain timing have no role but in Spiking Neural Networks (SNN) since spike's shape are same, then spike timing is very important.
- There are some models for implementing Neurons and Synapses (which are the main parts of biological neural networks) in VLSI, but since in a deep neural network there are too many connections (synapses) and also too many neurons, the power consuming is very important. Some of implementations try to reduce the power consuming in Neuromorphic hardware. One of the new and the state of the art technology which announced in 2008 by HP [13] and had shown very perfect ability in synapses implementation in VLSI with low power consuming, is *Memristor*.

### 1.1.2 The Proposed Approach

In this thesis considering the rate aspect of neural coding, we proposed an approach to overcome the first challenge of previous subsection. We proposed an abstract model for Leaky-Integrated and Fire (LIF) spiking model which allows us to use spiking neurons in a deep architecture called Deep Belief Networks (which will be introduced later) in an *online learning* manner. The final goal of this study is a *Neuromorphic Accelerator* for the proposed *Spiking Deep Belief Network*, but in this thesis we will not consider the hardware implementation and we talked about it only in the future works. Indeed providing a framework for utilizing Contrastive Divergence (CD) from Machine Learning domain in a Spiking Neural Network with Leaky Integrated-and-Fire (LIF) neurons, we have smoothed the way for the hardware implementation of the model. Finally using of Memristors is suggested to obtain a Hardware Accelerator for Deep Belief Networks.

Since the recent architecture of Spiking Neural Network dose not obey the Von-Neumann architecture, then the model can provide a low power model [14, 15]. Non-Von Neumann architecture leads to less instruction and data transferring from and to memory. Such an architecture which can compute in parallel is a *Neuromorphic Accelerator*. (Indeed we are interested in implementing a Neuromorphic Accelerator in future works using Memristor cross-bars.)

In following chapters we will indicate the *Learning* concept in chapter 2, then Artificial *Neural Networks* and some *Neuron Models* will be investigated and also we will address the mathematical and probabilistic basis. Then introducing *Restricted Boltzmann Machine (RBM)* we will address *Deep Belief Networks (DBN)*. In the following we will demonstrate some application of DBN. In chapter 3 at first we will discuss about *Spiking Neuron Models* and then we will propound our main challenge in using spiking neuron models in machine learning domain which is the subject of this thesis.

# Chapter 2

## A background in Neural Learning and Learning in Deep Belief Networks

# 2 A background in Neural Learning and Learning in Deep Belief Networks

Deep Belief Networks (DBN) as a kind of deep architectures, was proposed by Professor Geoffrey Hinton in 2006[10], stacking and training some layers of Restricted Boltzmann Machines. Training the stack of RBMs can be used in network initiation or for pre-training a supervised model of ANN. A RBM with *n* hidden units, is a *Morkov Random Field (MRF)* for joint distribution between hidden and observed variables. The RBM model, can be trained using approximation of *Stochastic Gradient Descent (SGD)*. The *Contrastive Divergence (CD),* which introduced in [16], is a very good stochastic approximation algorithm for estimating the values of $\frac{\partial \log(P(x))}{\partial \theta}$. Deep belief networks are generative stacked models in multilayer architecture. Using Contrastive Divergence algorithm, each observable data which passed though each layers of DBN, will be used as input value for next layer. Repeating this layering, we have trained stacked Restricted Boltzmann Machines.

Since the purpose of this thesis is making closer DBN from *Machine Learning (ML)* domain to *Spiking Model* in *Time Domain*, therefore in this chapter, we will explore learning issue and then discuss about *Neural Network* as the brain imitating tools in two point of views: first regarding the biological aspects (*Spiking Neural Networks*) and the second regarding mathematical modeling (*Artificial Neural Networks*) and we will have a look at common models of artificial neurons. In the following we will introduce the generative models and Deep Belief Networks.

## 2.1 Learning Algorithms

Computer programming for some applications is very difficult. For example it's too hard to program computers to detect some specific objects in an image or to predict some percept depended applications. Generally learning based methods will be suggested to deal with these cases. There are some various definitions for *Machine Learning (ML)*. The most popular one defined by Arthur Samuel is: "Field of study that gives computers the ability to learn without being explicitly programmed". Another more official and perfect definition proposed by Tom Mitchell as [17] :" A computer program is said to learn from experience

E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E". Most of machine learning algorithms are categorized as bellow[1]:

- Supervised Learning Algorithms
- Unsupervised Learning Algorithms

There is a category of methods that is generally a subset of Supervised Learning Algorithms and called *Semi-Supervised Learning Algorithms*. In semi-supervised learning methods learning process uses labeled and unlabeled data, which most of data may consist of unlabeled data. Interestingly it was shown training using data which some of them are labeled (and not all of them), in some cases, can offer fairly accurate answers.

### 2.1.1 Supervised Learning Algorithms

Mostly this methods are looking for relations between inputs and outputs (as labels for inputs). Learning in supervised manner uses a dataset of labeled elements as *Training Data*. To estimate the accuracy of the learning algorithm, we have to use another dataset as *Test Data*. Training data will be used as ordered pairs consist of data and favorable output (as label for data). The supervised method by studying these ordered pairs can find an appropriate relationship between inputs and outputs. Therefore using the gained relationship, the system can predicts outputs for new inputs. The most popular supervised algorithm, is *Back-Propagation* [18].

### 2.1.2 Unsupervised Learning Algorithms

In machine learning domain, the Unsupervised Learning Algorithms, have pointed the methods which trying to find the pattern and internal representation of data by exploring the unlabeled inputs. These methods are much closed to stochastic and probabilistic problems. Unsupervised Learning Algorithm is looking for the key features of data. Mostly *Deep Learning (DL)* algorithms are from this category.

## 2.2 Neural Networks: Learning from Neurons

The term of Neural Network indicates the efforts to find a mathematical demonstration of data processing in biological systems [3]. Indeed this term was used for many models. The

[1] There are some types of algorithms called *Recommender Systems* and *Reinforcement Learning* that will not be discussed in this thesis.

used algorithms are looking for a way for imitating the brain operation. The human knowledge about how brain works is increasing, but the gained knowledge from years ago until now, have been the foundation of achieving Artificial Neural Networks (ANN). Since the McCulloch and Pitts in 1943 [19] proposed their simple models for neuron until now some perfect and powerful models have been introduced by scientists and researchers. The primary proposed model in [19] is consisted of a simple binary model that implemented an *Activation Function (Thresholding Function)*. Introducing this model two filed of investigations in neural networks have been considered. The first category talk about the computational applications and the second one focused on the biological features of neural networks and the learning problem. Using and considering the neural network based algorithms, after publication of the *Perceptrons [20]* and studying the benefits and disadvantages of this model, was declined. Generalizing the weakness of neural networks to all types of neural networks, which was made by researchers, was the major result of the isolation of neural networks. As soon as the computers achieved more processing power, and new models of artificial neural networks was invented, neural network research found life again.

One of the important aspects in brain structure is using of parallel architecture and including too many connections between neurons. This architecture cases high speed in brain processing. Human brain consisted of about $10^{11}$ neurons that each neuron has about $10^{4}$ *Synaptic Connections* to other neurons. Figure 2-1, shows the structure of a biological neuron.

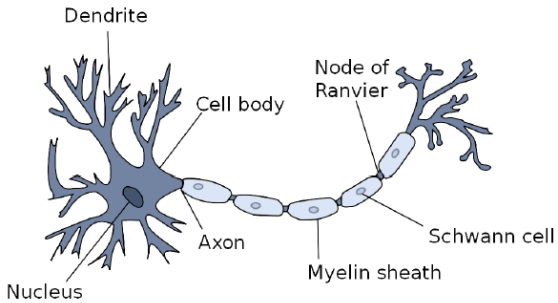


**Figure 2-1: Biological neuron structure [21]**

Using this density of neurons and connections, the parallel processing is possible. Neurons are the main part of neural networks and have some different kinds. Neurons can send and receive messages to and from other neurons and muscles. As was shown in Figure 2, there three main parts in any biological neuron:

1- *Cell body*: *Soma* or *Cell body*, is the cell core. This part can provide required energy for other parts and also is the processing center of a neuron. In simple models of neurons, this part is emulated using an adder circuit followed by an activation function.
2- *Dendrites*: They are strings from cell core branched to all sides. Dendrites are responsible for collecting neural stimulus from other neurons and transferring them to cell body.
3- *Axons*: They are pretty long neural fibers issue from cell body. They have smooth surface. Axons using their length, can transfer the processing results from cell body to other neurons. It is noticeable that in all types of neurons the length of Axons is no longer than the length of Dendrites. In *Sensory Neurons*, which are responsible for collecting the environmental stimulus such as light, the Dendrite's length is longer than Axon's length. This model of neurons transfer the messages from sensory receptors to brain. Another type of neurons used for transferring the electrical pulses and commands from brain and spinal cord to muscles and organs, have longer Axons. They called *Motor Neurons*.

   Neural Message passing used neural junction between one neuron's Axon and another neuron's Dendrite. This connection is not a direct connection. Indeed there is a gap in neural junction place and *Neurotransmitter* using electrical charges, transfer the messages. This gap is a very specific biological structure called *Synaptic Cleft* (Figure 2-2)and has very important role in learning process [22]. Each neuron can receive stimulus from its neighbors. The impact of each input is affected by the *Synaptic Weight*s. Actually the learning is the process of adjusting the *Synaptic Weights*. These weights will adapted in brain to have complete computation for various applications like object detection, organs controlling or natural language processing. Using the huge amount of neurons and their connections, too many Synaptic Weights impacted the computational process in a very short time.

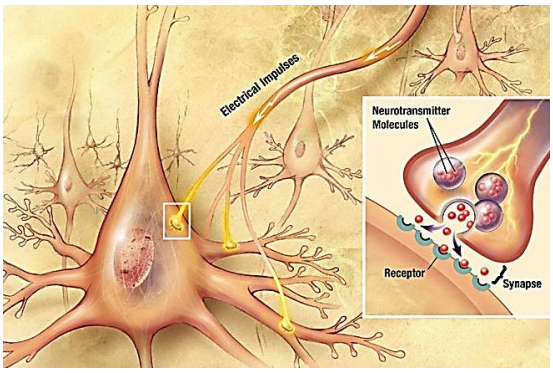


**Figure 2-2: Synapse as the neural junction [23]**

### 2.2.1 Two different viewpoints

As was discussed, the Neural Network, as a solver tool, indicates the methods for mathematical demonstration of brain process for learning and overcoming the problems. Too many approaches was proposed for using neural networks in computations. In 1997, Maass [24] proposed three generations to classification the neural network methods:

*The 1'st generation*: Using the McCulloch-Pitts neurons as the computational units. This models generally was classified as *Perceptron Neurons*. Usually this generation uses units with binary output states. *Multi-Layer Perceptron, Hopfield Model* and *Boltzmann Machines*, are some example of this generation.

*The 2'nd generation:* The neural networks which uses units with a continuous set of possible output values. They have a real values as output of each units which can be interpreted as the *firing rate* of each neuron. *Sigmoid Units* are the most general model of this generation which can be used in a network of neurons. This types of networks can support *Gradient Based* algorithms (e.g.: *Back-Propagation*), as a main characteristic of this generation.

*The 3'rd generation*: As was explained, the real biological neurons uses the *Spikes* (which are small electrical pulses) to communicate with each other. In this model of neurons the *prices spike times* or *firing times* are very important and have the main role in learning process in biological neurons. This generation which called *Spiking Neural Networks* are suitable for VLSI implementation.

In the following we discussed these three generation of models in two main category. The first one is *Artificial Neural Networks* which is consist of 1'st and 2'nd generation of neural networks and the second one is *Spiking Neural Networks* which mentioned as the 3'rd

generation of neural networks.

#### 2.2.1.1 Artificial Neural Networks (ANN)

In a simple model of an artificial neuron, Dendrites are considered to be the inputs of neuron and Axon are the neuron outputs. Accordingly we have a simple artificial neuron as the main computational unit in Artificial Neural Networks (Figure 2-3).

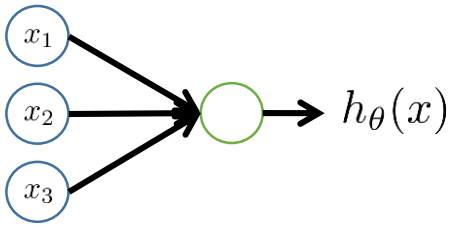


**Figure 2-3*: A simple model for Artificial Neuron***

Artificial Neurons are abstraction of real neurons. Generally these neurons receive inputs (from Dendrites) and generate the output (Axon) using add operator. Each Dendrite can multiply by Dendrite's weight value. Dendrite's weight values are stored in synaptic cleft and express the strangeness of relation between to neurons. This weights represent the intervention of each input to achieve a specific result. In biological neurons, the weight adjustment is made by adjusting the signaling transfer rate and using *excitatory* and *inhibitory* signals.

The adder unit in artificial neuron will followed by a nonlinear function called *activation* or *transfer function*. Generally, the transfer function is a *Sigmoid Function*, but other types of nonlinear functions such as *piecewise linear* and *step function* can be used.

The output of neuron (Axon), can be transferred to next layer or directly to output layer. In the following, we briefly introduce some common types of artificial neurons.

#### 2.2.1.2 Some simple artificial neurons

At the beginning of this section, we want to introduce the concept of *Transfer Function*. *Transfer Functions* will be used for simplification or optimization of neural network's features. These functions will be applied on the total weighted of inputs. In equation (2-1) u demonstrated the total weighted and $x_i$ are the element of X as the input vector and $w_i$ are the elements of W as the weigh vector:

$$u = \sum_{i=1}^{n} w_i x_i \qquad (2\text{-}1)$$

The function that will applied on u is the *transfer Function*. As we discussed before, the primary proposed model by McCulloch and Pitts [19], has a binary implementation which applied a binary transfer function on the total weighted using the comparison between the total weighted and a threshold value. If total weighted is bigger than the threshold the output is one and else is 0 (Figure 2-4).

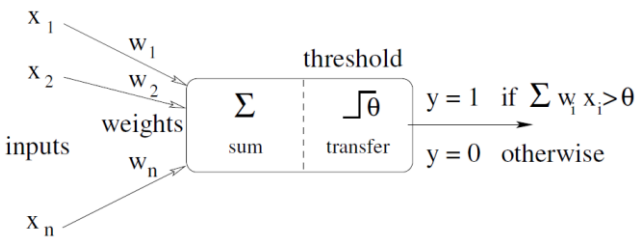


**Figure 2-4: The first mathematical model for artificial neuron [19]**

The first proposed model for artificial neuron (Figure 2-4) has the main aspects and properties of neurons [25]. How a model deal with the input vectors and the weight matrix, is the main difference between various proposed models. Considering the McCulloch and Pitts model, one of the most applicable type of the artificial neurons is based on sum of weighted inputs (can be explained using inner product of input vectors and the weight matrix:$< \boldsymbol{X}, \boldsymbol{W} >= \sum_{i=1}^{n} w_i x_i$). Using these sum of weighted inputs and applying some different *transfer functions*, some different type of artificial neurons will be obtained. The following models are the results of applying some different transfer functions:

**Linear Neuron Model:** In this model, the output is combination of weighted input which is biased using a constant value (Figure 2-5). The equation (2-2) can describe the behavior of this model.

$$y = b + \sum_{i=1}^{n} w_i x_i \qquad (2\text{-}2)$$

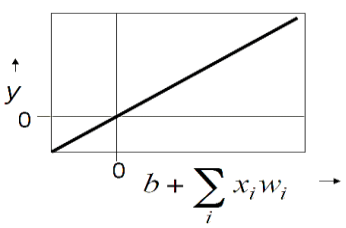


**Figure 2-5*: Linear Neuron Model***

**Binary Threshold Neuron Model:** Roughly speaking, it is the same model as McCulloch and Pitts model [19]. When the sum of weighted inputs is larger than a pre-defined threshold ($\theta$), the output can catch a specific value (for example logical one) (Figure 2-6). The equations for this transfer function is (2-3):

$$z = \sum_i x_i w_i$$
$$y = \begin{cases} 1 & if\ z \geq \theta \\ 0 & otherwise \end{cases} \qquad (2\text{-}3)$$

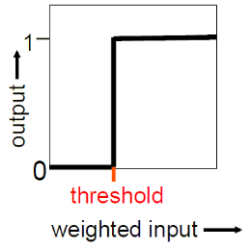


**Figure 2-6*: Binary Threshold Neuron Model***

Adding a bias value ($b$) to the sum of weighted inputs, we can rewrite the equation (2-3) as (2-4):

$$z = b + \sum_i x_i w_i$$
$$\theta = -b$$
$$y = \begin{cases} 1 & if\ z \geq b \\ 0 & otherwise \end{cases} \qquad (2\text{-}4)$$

This model of neuron is used in *perceptron* and too many other models.

**Rectified Linear or Linear Threshold Neuron Model:** In this model the output of the model will be shown as a non-linear function of sum of weighed inputs (2-5):

$$z = b + \sum_i x_i w_i$$
$$\theta = -b$$
$$y = \begin{cases} z & if\ z \geq b \\ 0 & otherwise \end{cases} \qquad (2\text{-}5)$$

As you can see in Figure 2-7, this model is a combination of *Linear Neuron Model* and Binary Threshold Neuron Model.

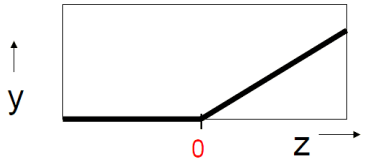


**Figure 2-7*: Rectified Linear Neuron Model***

In this model, the linear combination of inputs will be passed through a thresholding function and as a result the output is nonlinear function of inputs.

**Sigmoid Neuron Model:** This type of neurons are very important because they have continues output which supports the gradient-based learning rules. The output of this model is bounded real value which is a result of using *Logistic Function* as the transfer function. The *Sigmoid Function* (2-6) is an S-shape mathematical function.

$$z = b + \sum_i x_i w_i$$
$$y = \frac{1}{1 + e^{-z}} \qquad (2\text{-}6)$$

This function is bounded, differentiable and defined for all real input and has a positive derivative at each point (Figure 2-8).

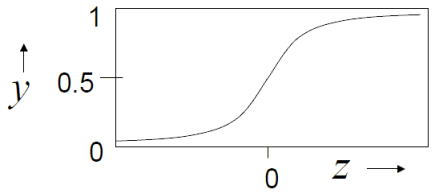


**Figure 2-8*: Sigmoid Transfer Function (Logistic Curve)***

**Stochastic Binary Neuron Model:** This model uses a transfer function same as *Logistic Function*, but the output of the transfer function is the probability of generating a spike in a short time window (2-7). The output of this type of unit is a real value between 0 and 1 (Figure 2-9).

$$z = b + \sum_i x_i w_i$$
$$p(s = 1) = \frac{1}{1 + e^{-z}} \qquad (2\text{-}7)$$

(S=1 means a spike is generated)

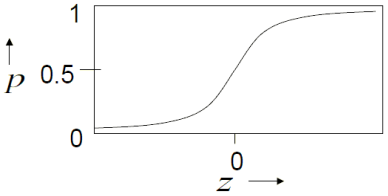


**Figure 2-9*: Stochastic Binary Neuron Model***

According to equation (2-7):

- If z is big and positive, then output is mostly one.

- If z is big and negative, then output is mostly zero.

This model can be used with *rectified* output. This means the output will be rectified with the probability of logistic function. In this situation output is a real value which is generated using Linear Rectified function, instead of binary values.

In this last model, if the summation of inputs is larger than zero (z>0), then the output can be interpreted as the spike generation rate, (but cannot show the spike generation times which is important in Spiking Neural models).

#### 2.2.1.3 Spiking Neural Networks (SNN)

*Spiking Neural Networks* or *SNN* are a class of neural networks using *Spikes* or *Action Potentials* for transferring and coding data and commands in biological inspired neural networks. SNN increased the level of realism in neural simulation. In SNN both of synapses and neurons are take into account for computational tasks. In traditional neural networks the computed values in neurons will be propagated in each cycle (or in each clock pulse of simulation), but in SNN each neuron fires only when the *membrane potential* crossing the *threshold* as a specific value. The generated spikes will be sent to other neurons which are connected to the source neuron. The receiving pulses can increase or decrease the membrane potential of the receiving neurons and accordingly can force the neurons to fire or prevent them from firing. Spiking Neurons usually will be explained using differential equations which can determine the neuron's state at the specific times.

Action Potentials (Spikes) rapidly rises and falls and can be generated as a temporal sequence called Spike Train. A typical Action Potential is depicted in Figure 2-11. As was explained, the neurons sends spikes across the synaptic junctions. The sending neuron, usually, is called presynaptic cell and the receiving neuron is called postsynaptic cell. The spikes consist of short electrical pulses and have typically a duration of 1 to 2 ms and amplitude about 100mv. Each spikes, individually, have same forms, since the spike's form have no role in information transferring. As it is explained in [26] by professor W.Gestner et al., the *number and the time of spikes* are important characteristics in neural communications.

When a neuron generates a spike, even very strong input cannot force it to spike immediately after the first one. The minimum time between two spikes is defined as the *absolute refractory period* which in this period of time the neuron cannot generate any

spikes.

Synaptic junctions usually are chemical spaces between pre- and postsynaptic cells called also s*ynaptic cleft*. Arriving an action potential at a synapse, starts a biological processes that lead to *Neurotransmitter* releasing from presynaptic into synaptic cleft (Figure 2-10). When the neurotransmitters reached the postsynaptic cell, the specific channels will be opened and the ions from the extracellular fluid to flow into the cell. This ion influx can change the membrane potential at the postsynaptic side. If the amplitude of the membrane potential be large enough, the postsynaptic cell can generate a new spike [26].

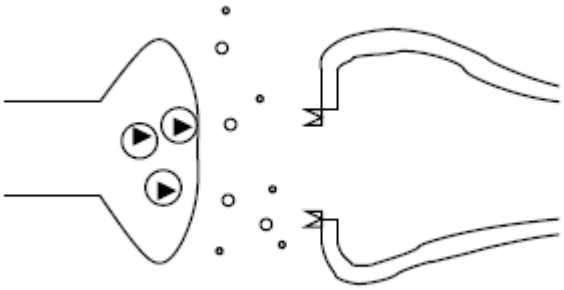

**Figure 2-10: Presynaptic neurons release neurotransmitter molecules into synaptic space [27]**

Indeed the membrane potential is the potential difference between the inside and outside of a neuron's cell. Usually for the membrane potential $u(t)$ will be used (which emphasizes time of any level of potential). When there is no input, the neuron has a constant level of potential called *resting potential* ($u_{rest}$). After any spike the potential will be decayed back to $u_{rest}$ (has negative polarization about -65mV) [28].

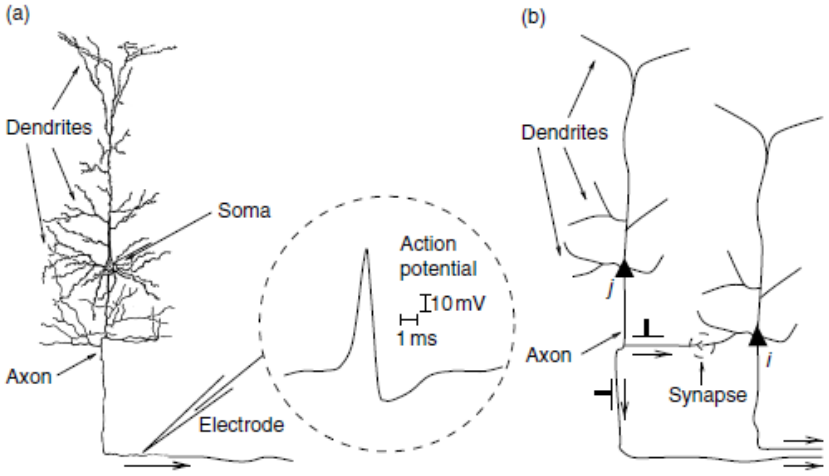


**Figure 2-11: Action potentials travel from presynaptic cells to postsynaptic cells [26]**

When a spike is arrived the membrane potential will be increased or decreased. The positive changes are made by excitatory synapses and the negative effects made by inhibitory synapses. Depolarization happened when an excitatory input is received and reduces the negative polarization and when an input increases the negative polarization hyperpolarization will be occurred.

## 2.3 Deep Learning

Deep Learning methods are focused on learning from hierarchical features that produced by low to high levels of abstraction. There are some challenges in learning using deep architectures. One of the most important problem in learning process using deep networks, is the strong dependency between the layer's parameters.

Basically deep learning indicates training using some layers of abstraction and representation. This style of learning can improve the machine perception of voices, images and etc. As is mentioned in [29], some various algorithms for training these networks are proposed in several papers. These algorithms will start learning process commonly, using a greedy and layer wised pre-training process that can initiate the network. The learning process will follow by a supervised fine-tuning method. Each layer will be trained using unsupervised pre-training algorithm. Through the learning process, layers can learn a nonlinear mapping from their input (that can be the output from previous layer). In fine-tuning phase, as the final step, the results from deep architecture will be sued by a gradient based supervised optimization algorithm. In [29] it was shown that using very large dataset, unsupervised pre-training can reduce the classification error.

The proposed algorithms in 2006 are layer-wised and suitable for deep architectures. One of the most perfect algorithm proposed by Geoffrey Hinton for training Deep Belief Networks is a greedy and layer wised method which trains entire of the network. The implemented architecture for DBNs is consisted of multiple stacked Restricted Boltzmann Machines as the layers of the network. Professor Hinton and his team was shown the efficiency and preference of this algorithm in dimension reduction [30] and classification and inferring [10] in machine learning domain.

In this study we are looking for a way to use spiking neurons as the computational units in deep architecture proposed by professor Hinton for deep learning called *Deep Belief Networks*. Therefor it is necessary to study this proposed mathematical algorithm in detail.

In this way we have to introduce some terms and models such as *Probabilistic Graphical Models (PGM), Markov Random Fields (MRF), and Restricted Boltzmann Machine (RBM).*

### 2.3.1 Probabilistic Graphical Models (PGM)

*Mathematical model*, is a system description using mathematical syntax and semantic. *Mathematical models* are applicable in natural sciences such as Physics, Biology, Geometry, Social Sciences and Political Sciences and also in engineering fields such as Computer Sciences and Artificial Intelligence. A model can explain a system and help to predict the behavior of the system dealing with various components [31].

The Mathematical Models, generally, are composed of variables and relationships, which the relationships can be expressed using operators (e.g.: algebra operators, functions and etc.) and variables are abstraction of the system parameters. These models according to their common properties can be classified based on specific aspect. One of these classifications, classify deterministic and probabilistic models. In a deterministic model, each set of system states can be determined using model parameters and previous set of states. Therefore a deterministic model, for a given set of initial states, always has the same answer. Conversely, in a stochastic model (usually called statistical model), system states cannot be determined uniquely but rather probability distributions can expressed the states of the system [32]. The *Graphical Models* or *Probabilistic Graphical Models (PGM)* are one the tools for statistical models analysis, composed of Graph and Probability theory [3]. In a *Probabilistic Graphical Model*, a graph can expresses the conditional dependency of random variables. These models are useable in probability theory, statistics and machine learning [33]. The *Probabilistic Graphical Models* are a kind of *Probabilistic Models* which are applicable in various fields of researching. They provide a mathematical framework for understanding the relationships between the computational methods based on networks and in particular for understanding neural networks algorithms [34]. Using *Probabilistic Graphical Models* to demonstrate probabilistic distributions, can help in statistical inference and provide a powerful framework for machine learning and pattern recognition [35]. Considering the directions of edges between nodes, Graphical Models can be divided into *Directed* or *Undirected Graphical Models.* Generally the Directed Graphical Models are known as *Bayesian Networks* and Undirected Graphical models are called *Markov Random Fields (MRF).*

### 2.3.2 Generative Models

*Generative Models* are a subset of probabilistic models. Since by sampling from the joint distribution it is possible to generate some examples of any input vector, these models are called Generative Models [36]. A generative model can specifies a joint probability distribution over observation and label sequences. Generative models by modeling the observable data distribution to generate samples, are useable in Machine Learning. Using the estimated distribution over observed data, the conditional distribution can be evaluated to predict the labels for new data (The conditional distribution uses Bayes' rule).

Gaussian Mixture Model (GMM), Hidden Markov Model (HMM), Naive Bayes and Restricted Boltzmann Machine (RBM) are some examples of Generative Models.

## 2.4 Deep Belief Networks (DBN)

*Deep Belief Networks (DBN)* as very powerful tools in Machine Learning, have become the state of the art in many applications. They are generative models consisted of multi-layers of *Restricted Boltzmann Machine (RBM)* [37]. RBM is the main building blocks of DBN and other types of deep learning tools. RBMs can be assumed as *Recurrent Neural Networks (RNN)* which are a class of ANN. In RNN the edges between artificial neurons can form a cyclic path in the network graph. Each Restricted Boltzmann Machine has two layers. The first layer is a layer of visible units (input/visible neurons). The inputs from environment can be placed at this units as the observed data, also in a multi-layer (stacked) model of RBMs, the inputs can be generated by previous RBM. The second layer is consisted of hidden units (output/hidden neurons). This recent layer is responsible for modeling the dependency relation between the observed data. For example in a system which is prepared for image processing, the observed data are pixels of each images. The vector of these pixels will be used in input layer. The second layer can extract the dependency relation between the pixels. This layer can be assumed as a nonlinear feature extractor. In RBMs, as a RNN, each unit in first layer is connected to units in second layer, but there is no visible-visible or hidden-hidden connections. Indeed, the term *"Restricted"*, refers to eliminating these in-layer connections from original *Boltzmann Machine* which is a fully recurrent network.

As was mentioned, RBMs are generative graphical models, which can be used for representing the dominant distribution over observed data using sampling the learned distribution. Employing the RBMs in a multi-layer structure can provide a wide verity of machine learning tasks such as classification and image processing [38-41], voice and audio

analysis applications [42-44] and dimensionality reduction [30]. In this section at first we introduce *Boltzmann Machine* and *Restricted Boltzmann Machine* and briefly we explore learning process and weight updating in RBMs. Finally we can introduce the proposed model in [10] that called *Deep Belief Networks.*

### 2.4.1 Boltzmann Machine and Restricted Boltzmann Machine (RBM)

Restricted Boltzmann Machines as a model of Artificial Neural Networks, have been driven from Boltzmann Machines, consisted of binary stochastic units connected using bidirectional edges. They can represent a probabilistic distribution. Boltzmann Machines can learn the basic features of an unknown distribution using observed data. The observed data are used as the training data. Basically, training in a Boltzmann Machine, is parameters adjustment so the machine can fits an estimated distribution over observed data. Generally, training in Boltzmann Machine, as fully recurrent network, involved too complex computations. Therefore, applying some restrictions to Boltzmann Machine topology, leads to a less complex structure called *Restricted Boltzmann Machine*. Looking Boltzmann Machine in detail [45], we have a parallel structure consisted of basic elements called *Units* which are connected using bidirectional connections. Each *Units*, always, are in state *On* or *Off*. The *state of units* and their connection's weights will be adapted as a probabilistic function of the neighbor units.

The structure of Boltzmann Machine is related to proposed structure in 1982 by John Hopfield [45]. In Boltzmann Machine similar to the proposed model, each *Global State* (the state of entire network), can be expressed as value called *Energy of State*. This naming is because of the analogy with physical systems. The Hopfield model is driven from thermodynamics systems and can be quantified through equilibrium energy. Considering physical systems, a stable state for system is when reaching thermal equilibrium [46]. This state of system is stable because in thermal equilibrium, the probability distribution over configurations settles down. Each binary states of entire network, called the *network configuration*. Global network configuration, is related to its energy by Boltzmann distribution. The global configuration is a joint configuration over both visible and hidden units state which depends on their probabilities (joint probability over visible and state of hidden units, expressed as $p(\boldsymbol{v}, \boldsymbol{h})$). In physical systems, stationary distribution called *thermal equilibrium*. The proposed model find the minimum distribution probability to leads system settle in a thermal equilibrium state. Changing external thermal state, in physical

system, can change thermal equilibrium point, this means the internal state of the system will changed to settle to a stable state.

As was mentioned, the energies of joint configurations are related to their probabilities ($p(\boldsymbol{v}, \boldsymbol{h}) \propto e^{-E(\boldsymbol{v},\boldsymbol{h})}$). In Boltzmann Machine the total energy of any configuration can be expressed as (2-8):

$$E(\boldsymbol{X}, \boldsymbol{W}) = -\frac{1}{2}\sum_i \sum_j w_i x_i x_j - \sum_i b_i x_i \qquad (2\text{-}8)$$

Where:

E: The energy of the network

**X**: The state of all units in the network ($\boldsymbol{X} = \{x_1, x_2, \dots, x_N\}$)

**W**: The set of weights in the network

$w_{ij}$: The weight between unit i and unit j

$b_i$: The bias value for unit i

Here we are looking for a configuration of system which the network will converged to, over all possible configurations. Thermal equilibrium for a physical system will reached when the internal temperature and the external one, are same. Similarly, in a data model, changing the input values means new configuration. In such a case, the trained Boltzmann Machine try to represent the input vector of data. This means system is trying to reach internal state same as external state and this means equilibrium.

In Boltzmann Machine, which uses Boltzmann distribution, all the units are symmetrically connected. Since, we want to have two sets of units, discriminated as input and output units, we break **X** up into two vectors **V** consisting the visible (input) units and **h** for hidden units. Now we can rewrite the energy function (2-9):

$$E(\boldsymbol{X}, \boldsymbol{W}) = E\big((\boldsymbol{v}, \boldsymbol{h}), \boldsymbol{W}\big) \qquad (2\text{-}9)$$

Considering the Boltzmann distribution, as the distribution over Boltzmann Machine units, and the relation between energy function and their probabilities, the joint probability of **v** and **h** can be expressed as (2-10) :

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W})} \qquad (2\text{-}10)$$

When:

$$Z = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W})} = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} e^{-E((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W})} \qquad (2\text{-}11)$$

Marginalizing out the distribution over all hidden state **h**, will leads us to probability over visible units (**v**) (2-12):

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)} = \frac{\sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}} \qquad (2\text{-}12)$$

Generally, unsupervised learning indicates learning the main features of an unknown distribution like q, using the samples of q [47]. Any distribution can be used knowing its parameters. Assuming the graphical model is known and the energy function is expressed as a function of **w**, then unsupervised learning over observed data is involved finding and adjusting **w** parameters so that using these estimated parameters, system can generate closed samples to observed data (training data). Let **X** as the training data. Suppose that, the samples are driven from same unknown distribution and all the samples are independent.

The *likelihood function*, as an approximation of observed data (***X***), can be expressed using the distribution parameters (***W***), considering the independency condition for the observed data (2-13):

$$\mathcal{L}(\boldsymbol{W}|\boldsymbol{X}) = p(\boldsymbol{X}|\boldsymbol{W}) = \prod_{i=1}^{l} p(x_i|\boldsymbol{w}) \qquad (2\text{-}13)$$

Then the *likelihood function* ($\mathcal{L}$: W $\rightarrow$ $\mathbb{R}$) regarding observed data ***X***, is responsible for mapping ***W*** parameters from W parameters space to $\mathcal{L}(\boldsymbol{w}|\boldsymbol{X})$. The value of *likelihood function*, is equal to probability of generating observed data using founded parameters, which means the conditional probability of observed data given calculated W parameters. It is clear that, the bigger value for this probability means better approximation of target distribution parameters. Since *we are looking for a method to find the maximum value for this probability*. Basically, *Maximum-likelihood Estimation (MLE)*, is a method of estimating the parameters of a statistical model given data [48]. *MLE* is looking for statistical model parameters which lead to most similarity between estimated distribution and the unknown distribution. It means, the aim is to finding parameters using them someone can generates samples similar to observed data. In this way, the learning process is reduced to find the model parameters maximizing the likelihood function over observed data. Because the logarithm is a monotonically increasing function of its argument, maximization of the log of a function is equivalent to maximization of the function itself [3]. Therefore,

likelihood maximization is equal to maximizing *log-likelihood* (2-14):

$$\ln \mathcal{L}(\boldsymbol{W}|\boldsymbol{X}) = \ln \prod_{i=1}^{l} p(x_i|\boldsymbol{W}) = \sum_{i=1}^{l} \ln p(x_i|\boldsymbol{W}) \qquad (2\text{-}14)$$

In practical applications using log of likelihood can simplifies the mathematical analysis and help numerically.

Generally, for Boltzmann Distribution (Gibbs Distribution), the *Analytical Solution* to finding the parameters for maximizing the likelihood function, is impossible [47]. Therefore the *Numerical Solution*, such as *Gradient Ascent* are preferred. As was stated, maximizing the likelihood function means maximizing the probability of *observed data (training data)* generation using the estimated parameters, which means P as the estimated distribution, is similar to Q as an unknown distribution which generates the observed data. The similarity can be measured using some approaches. *Kullback–Leibler divergence (KL divergence)*, is a measure of the difference between two distributions P and Q [49]. If P and Q are identical, the KL measurement returns zero, otherwise it is a positive number. It can be proved maximizing log-likelihood is equal to minimizing KL divergence [10].

According to equations (2-12) and (2-13), for the training data presented at visible units we have (2-15) and (2-16):

$$\ln \mathcal{L}(\boldsymbol{W}|\boldsymbol{v}) = \ln p(\boldsymbol{v}|\boldsymbol{W}) = \ln p(\boldsymbol{v}) = \ln \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) \qquad (2\text{-}15)$$

$$\ln \mathcal{L}(\boldsymbol{W}|\boldsymbol{v}) = \ln \frac{\sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}} = \ln \sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)} - \ln \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)} \qquad (2\text{-}16)$$

To maximizing the log-probability of generating observed data (visible vectors) we can use gradient based method (instead of using analytical methods).

Doing gradient on (2-16) over **W** parameters (weights of the network) we have (2-17):

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{W}|v)}{\partial \boldsymbol{W}} = \frac{\partial}{\partial x}\left(\ln \sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}\right) - \frac{\partial}{\partial x}\left(\ln \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}\right) \quad (2\text{-}17)$$

$$= -\frac{1}{\sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}} \sum_{\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)} \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}}$$

$$+ \frac{1}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}} \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)} \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}}$$

With respect to (2-12) we can rewrite (2-17) as (2-18) [47]:

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{W}|v)}{\partial \boldsymbol{W}} = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}} + \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}} \quad (2\text{-}18)$$

When:

$$p(\boldsymbol{h}|\boldsymbol{v}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} = \frac{\frac{1}{Z} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \quad (2\text{-}19)$$

Considering the definition of *Expectation*, (2-18) can be written as the difference between expectation of *Energy Function* when model is generating samples on its own (using trained parameters and internal representation of trained data) and *Energy Function* when visible units clamped to the data (2-20):

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{W}|v)}{\partial \boldsymbol{W}} = \left(\left\langle \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}} \right\rangle_{model} - \left\langle \frac{\partial E\left((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\right)}{\partial \boldsymbol{W}} \right\rangle_{data}\right) \quad (2\text{-}20)$$

Where, $\langle . \rangle_{model}$ is an expected value over *equilibrium distribution* of Boltzmann Machine. Using the gradient of log-likelihood (2-20), in a gradient-based optimization algorithm such as *Gradient Ascent*, optimized set of weights for modeling the *hidden distribution*, can be achieved. To perform *Gradient Ascent* in the log probability, the Boltzmann machine would generate the observed data when sampling from its *equilibrium distribution*. Also using hidden units, $\langle . \rangle_{data}$ is the average, over all data vectors, when a data vector is clamped on the visible units and the hidden units are repeatedly updated until they reach equilibrium with the clamped data vector.

As is shown in (2-18), the gradient can be written as a sum of the corresponding two terms. Considering the binary state of units, we have to sum over all possible state of binary unit in visible and hidden layers. To avoid this time consuming calculations, we can estimate the gradient using the standard way called *Markov-Chain Mont-Carlo (MCMC)*. *Gibbs*

*Sampling* is one then *MCMC* sampling methods which has a wide variety of applications specifically in *Bayesian inference* and *Bayesian Belief Networks*. When direct sampling from a distribution is difficult, performing *Gibbs sampling* as a Markov-Chain Monte-Carlo algorithm for obtaining samples of observed data (approximated from a distribution such as joint probability distribution of two or more random variables or marginal distribution of one of the variables) [50].

### 2.4.2 Log-likelihood estimation and learning in RBM

Estimating using sampling method is possible when network has been settled to an equilibrium state, which is a long time process. Hidden units sampling can be done after updating phase and when the hidden unit are in stable state. In Boltzmann Machine, the time for settling to an equilibrium state, is exponentially depended on the number of units. Since, restricted model of Boltzmann Machine (were initially invented under the name Harmonium [37]) was used in which there no visible-visible and hidden-hidden connections (Figure 2-12).

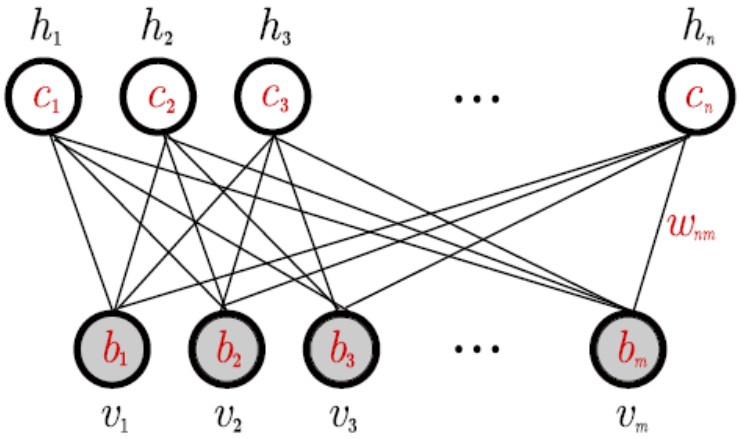


**Figure 2-12: Network graph for an RBM with m visible and n hidden units [47]**

As it is depicted in Figure 2-12, $c_i$s and $b_i$s are biases for hidden and visible units. So we can rewrite *Energy Function* as (2-21):

$$E(\boldsymbol{X}, \boldsymbol{W}) = -\sum_i \sum_j w_{ij} h_i v_j - \sum_j b_j v_j - \sum_i c_i h_i \qquad (2\text{-}21)$$

With respect to elimination of visible-visible and hidden-hidden connections, therefore can be concluded for each given visible vector, each hidden units are independent and can be updated independently in parallel. Also it is same for visible units.

According to (2-20) gradient of likelihood function for Restricted Boltzmann Machine

(RBM), can be expressed as (2-22):

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{W}|v)}{\partial \boldsymbol{W}} = \left( \left\langle \frac{\partial E\big((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\big)}{\partial \boldsymbol{W}} \right\rangle_{model} - \left\langle \frac{\partial E\big((\boldsymbol{v},\boldsymbol{h}),\boldsymbol{W}\big)}{\partial \boldsymbol{W}} \right\rangle_{data} \right) \qquad (2\text{-}22)$$

Or briefly (after performing derivation) as (2-23):

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{W}|v)}{\partial W_{ij}} = \left( \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} \right) \qquad (2\text{-}23)$$

To estimate the likelihood function gradient in this reduced (restricted) model, using MCMC , sampling process required to be iterated for several times. Gibbs sampling algorithm, as a simplified version of MCMC, is popular for generating samples form an unknown distribution over given samples. The main idea in Gibbs sampling method to update variables is based on using conditional distribution over other states of variables. Independency between units in layers of RBM, can simplified Gibbs sampling [51]. In RBM, instead of sequentially sampling from variables, all units in same layer can be sampled concurrently. This method which called *Blocked Gibbs Sampling*, can be accomplished in two phases: at first, the **h** vector, consisting of state of hidden units, will be updated according to $p(\boldsymbol{h}|\boldsymbol{v})$. In next phase visible units (collected as visible vector **v**), will be updated with respect to $p(\boldsymbol{v}|\boldsymbol{h})$, consequently.

Generally sampling methods, need to be repeated for many times until samples can be driven from an equilibrium state of network. Ideally, sampling process must to be repeated infinitely, which caused too difficulties in finding $\langle v_i h_j \rangle_{model}$. In Figure 2-13 the iterated Gibbs sampling to update hidden and visible units, is illustrated. Each sampling iteration leads to update all units in hidden units according to state of visible units and then updating visible units using state of hidden vector. This updating chain, is started after initialization of visible units with training vectors and can be repeated infinitely (t=infinity).

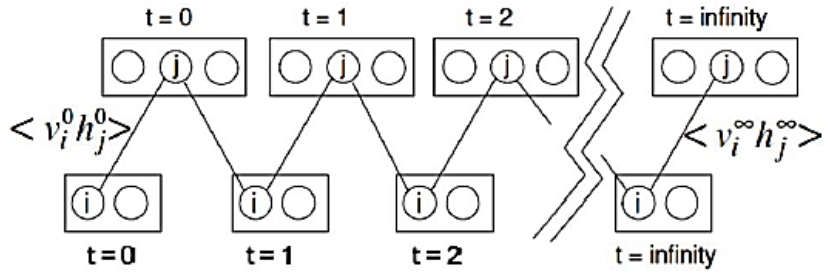


**Figure 2-13: Gibbs sampling [10]**

Practically, to achieve a reasonable estimation, Gibbs sampling can be repeated for too

many times (not infinitely). Using this method for RBMs consisting of too many visible and hidden units, is too slow. Professor Hinton, proposed a method to find maximum of likelihood function and estimating acceptable value of likelihood gradient [16]. The main innovation in Hinton's method, which called *Contrastive Divergence*, is initializing Gibbs updating chain using training data and iterating sampling for limited times (Figure 2-14). Clamping training data to visible units, with respect to independency between units of each layer, parallel sampling from hidden units is possible. With sampling hidden units, distribution of sampled data for both of visible and hidden layers, are available. In the next phase, using the samples of hidden units, we can *reconstruct* the training data. This *reconstruction phase*, generates a distribution over visible units called *model distribution.* In recent phase, system can generates a *model* of observed data, which is the network *representation* of observed data. More precision in weights adjustment, can caused to more similarity between *model distribution* and *data distribution* and the network can *reconstruct (represent)* observed data more precisely. The original proposed algorithm for pure Boltzmann Machine, because of the recurrent network, was too time-consuming. In [52] Peterson and Carsten have presented a *Mean Field* based method (MFBM), to approximate the stochastic correlation measurement in pure Boltzmann Machine. Indeed MF by providing an average value for a complex network of stochastic units, can reduced the problem size .Using this approximation the Boltzmann Machine has been able to be learned (the major steps of MFBM: Clamping phase, Free running and Updating) with less computational cost. This method proposed instead of computing $<S_iS_j>$ (where *Si* and *Sj* are the unit's state), one can use $V_iV_j$, (where $V_i$ *and* $V_j$ are the corresponding approximated MF values for $<S_i>$ and $<S_j>$). On the other hand, not only the method has been proposed for pure Boltzmann Machine, but also has required to run for a certain time. In [53] using same method for Mean Field approximation of distributions correlation (as sigmoid function), the method has been successfully tested on some machine learning tasks using RBM. In fact this version of Boltzmann Machine which is used in deep architecture not only is restricted in connections, but also using approximation of stochastic behavior of units (by MF) and running the chain only for one-step, leads to less computational costs.

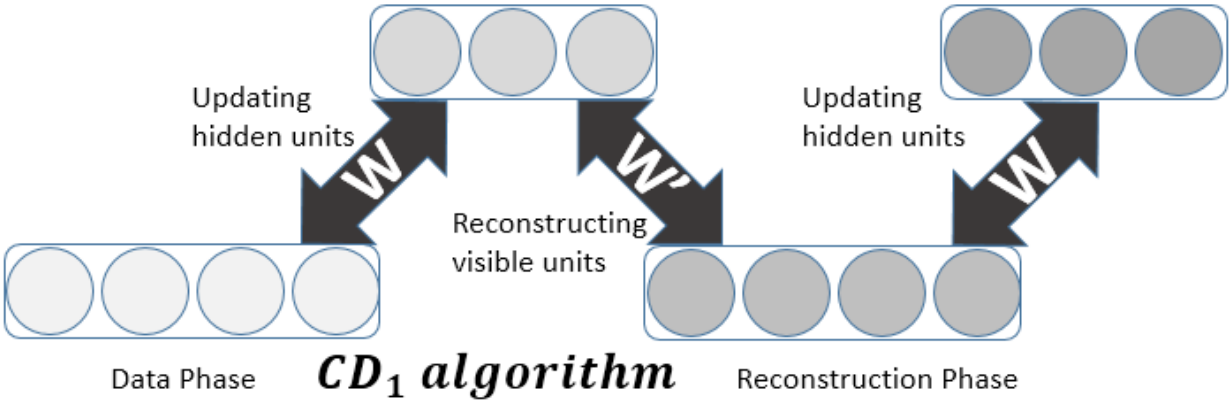


**Figure 2-14*: Contrastive Divergence (CD) algorithm***

Running *Contrastive Divergence (CD)* for *k* times ($CD_k$), follows very simple updating algorithm started from initializing visible units using training data ($\boldsymbol{v}^{(0)}$). In each iteration, $\boldsymbol{h}^{(t)}$, the state of hidden units in t-th iteration, is achievable using $p(\boldsymbol{h}|\boldsymbol{v}^{(t)})$ and $\boldsymbol{v}^{(t+1)}$ can be driven using $p(\boldsymbol{v}|\boldsymbol{h}^{(t)})$.

According to (2-18), $CD_k$ can be expressed as (2-24) [47]:

$$CD_k(\boldsymbol{W}, \boldsymbol{v}^{(0)}) = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(0)}) \frac{\partial E\left((\boldsymbol{v}^{(0)}, \boldsymbol{h}), \boldsymbol{W}\right)}{\partial \boldsymbol{W}} + \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(k)}) \frac{\partial E\left((\boldsymbol{v}^{(k)}, \boldsymbol{h}), \boldsymbol{W}\right)}{\partial \boldsymbol{W}} \quad (2\text{-}24)$$

The proposed structure for RBM is consisted of a network of *Binary Stochastic* neurons arrange in visible and hidden layer and visible units are symmetrically connected to hidden units using synapses which is illustrated as the weights assigned to edges (connections). Conditional probability of seeing '1' for each units (artificial neurons) can be interpreted as the *firing rate* generated through sigmoid function ($\sigma(x) = 1/(1 + e^{-x})$). Therefore the probability of '1' for each hidden unit ($h_j$), considering the stats of visible units ($\boldsymbol{v}$), can be written as (2-25):

$$p(h_j = 1|\boldsymbol{v}) = \sigma(c_j + \sum_{i=1}^{m} v_i w_{ij}) \quad (2\text{-}25)$$

Consequently for visible units we have (2-26):

$$p(v_i = 1|\boldsymbol{h}) = \sigma(b_i + \sum_{j=1}^{n} h_j w_{ij}) \quad (2\text{-}26)$$

Therefore, using $v_i$ and $h_j$ according to (2-25) and (2-26), and with respect to (2-23),

gradient estimation can be performed and $\Delta w_{ij}$ as the updating step in optimization algorithm is (2-27) [54]:

$$\Delta w_{ij} = \eta(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}) \quad (2\text{-}27)$$

To update biases same approach can be used. Then we have equations (2-28) and (2-29) for updating visible and hidden unit's biases:

$$\Delta b_i = \eta(\langle v_i \rangle_{data} - \langle v_i \rangle_{model}) \quad (2\text{-}28)$$

$$\Delta c_j = \eta(\langle h_j \rangle_{data} - \langle h_j \rangle_{model}) \quad (2\text{-}29)$$

To have complete updating iteration in *Contrastive Divergence* ($CD_1$) we have to use training data vectors at visible units, then using (2-25) states of hidden units can be determined. Using (2-26) in next step can update the states of visible units and reconstruct the data vector. This procedure can be iterated for many times, but Hinton has shown that $CD_1$, can estimate gradient of likelihood perfectly [55]. With respect to *Contrastive Divergence*, some faster and optimized algorithms was proposed. For example Tieleman proposed *Persistent Contrastive Divergence (PCD)* [56] and *Fast PCD* [57] that can achieve better learning by estimating the statistics of model using *"fantasy particles"*. *Fantasy particles*, means the model distribution persist across training iterations [57].

### 2.4.3 Learning in Deep Belief Networks

*Deep Belief Networks (DBN)* are probabilistic generative models composing of multiple layers of stochastic, latent variables. Generally these variables are have binary value and called *hidden units* or *feature detectors*. The main building block in deep belief networks, are RBMs. Using a trained RBM and adding one more layer on top the trained RBM, we can use the hidden layer of first RBM as the visible layer of new RBM. New RBM is composed of the new added top layer as its hidden layer and the hidden layer from trained RBM as its visible layer. The stages of adding new layer to RBM is illustrated in Figure 2-15:

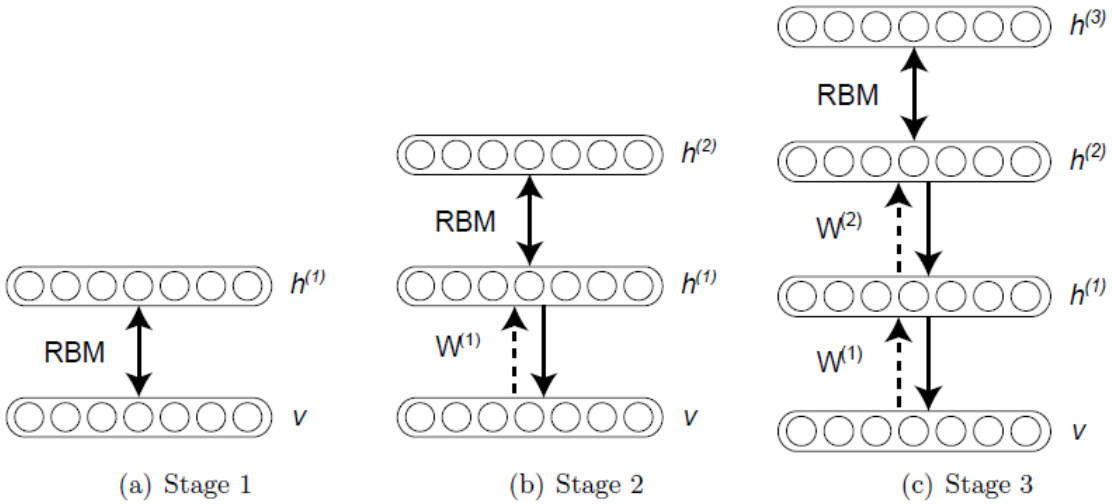


**Figure 2-15: Adding new hidden layer on top of an RBM [58]**

At first stage (Stage 1), an RBM will be trained. In this stage data can be passed through both back and forward between visible and hidden layers. In stage 2 and when a new layer is added, $h^{(1)}$ will be used as the visible layer for new RBM. In this stage the learning weights in the first RBM have no update. This is equivalent to learning another RBM, using *posterior distribution* of $h^{(1)}$ as data.

In this deep structure, first RBM will trained a layer of features that receive input directly from data vectors. Then the second RBM treat the trained features as data vectors and the recent RBM can learn features of features. In stages 2, the data vectors just passed through the weights of first RBM and then will be presented to new RBM. It is same for other layers. This process can be repeated and in each stage the upper hidden layer exploiting of the lower layers extracted features, can represent higher level of abstraction. This level of abstraction can help to have more precise classifications, too [29]. The proposed structure by Hinton is depicted in Figure 2-16. this structure uses a *greedy layer wise* algorithm and has some properties [10]:

1- There is a fast, greedy learning algorithm that can find a fairly good set of parameters quickly, even in deep networks with millions of parameters and many hidden layers.
2- The learning algorithm is unsupervised but can be applied to labeled data by learning a model that generates both the label and the data.
3- There is a fine-tuning algorithm that learns an excellent generative model which outperforms discriminative methods on the MNIST database of hand-written digits.

4- The generative model makes it easy to interpret the distributed representations in the deep hidden layers.
5- The inference required for forming a percept is both fast and accurate.
6- The learning algorithm is local: adjustments to a synapse strength depend only on the states of the presynaptic and post-synaptic neuron.
7- The communication is simple: neurons only need to communicate their stochastic binary states.

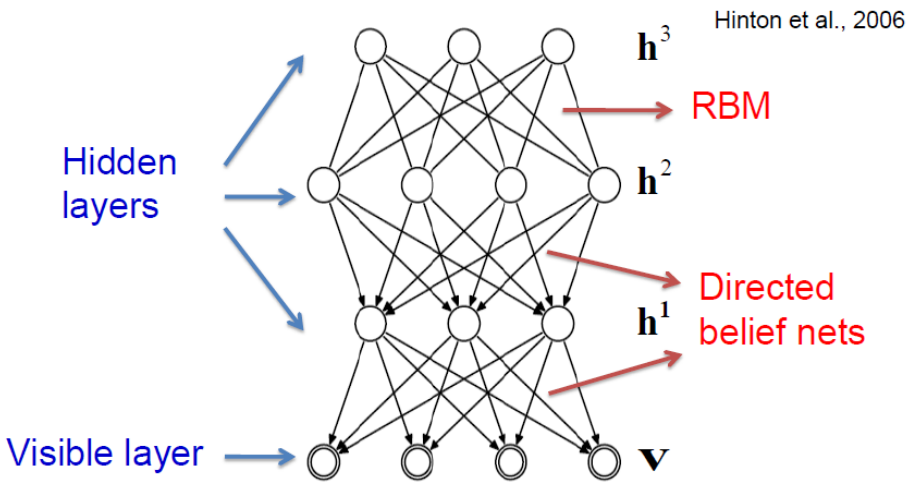


**Figure 2-16: DBN architecture [10]**

The abstraction of Hinton's algorithm can be expressed as bellow [10, 59]:

1- Learn weights ($\mathbf{W^1}$) and other parameter in first Restricted Boltzmann Machine, suing input vectors **v** and hidden units $\mathbf{h^1}$ (Figure 2-17).
2- Add a new layer (new top layer) on top of the recent trained RBM. Freezing $\mathbf{W^1}$ and other parameters for old RBM, sample $\mathbf{h^1}$ as input vectors for new RBM according to $Q(\boldsymbol{h^1}|\boldsymbol{v}) = p(\boldsymbol{h^1}|\boldsymbol{v};\boldsymbol{W^1})$ and calculate new RBM weights ($\mathbf{W^2}$) and parameters (Figure 2-18).
3- Add a new layer (new top layer) on top of the recent trained RBM. Freezing $\mathbf{W^2}$ and other parameters for old RBM, sample $\mathbf{h^2}$ as input vectors for new RBM according to $Q(\boldsymbol{h^2}|\boldsymbol{h^1}) = p(\boldsymbol{h^2}|\boldsymbol{h^1};\boldsymbol{W^2})$ and calculate new RBM weights ($\mathbf{W^3}$) and parameters (Figure 2-19).
4- Do same for other new layers.

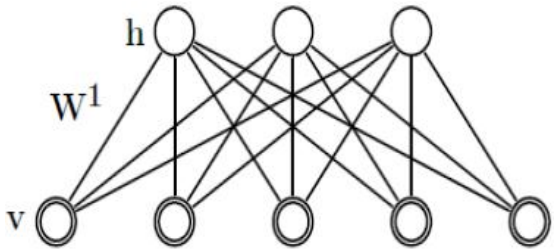


**Figure 2-17*: Training first RBM in DBN structure***

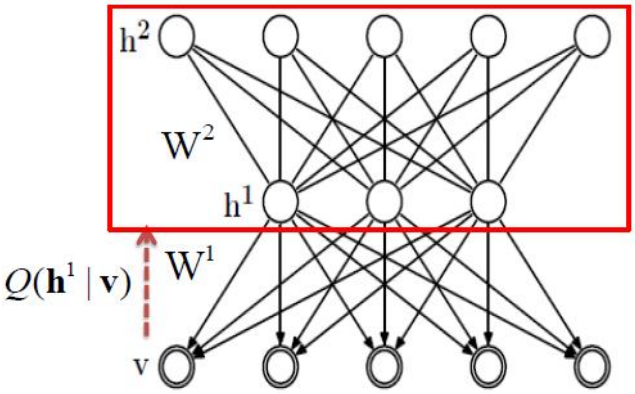


**Figure 2-18: Adding a new top hidden layer, freezing firts RBM weights and training the second RBM in DBN structure**

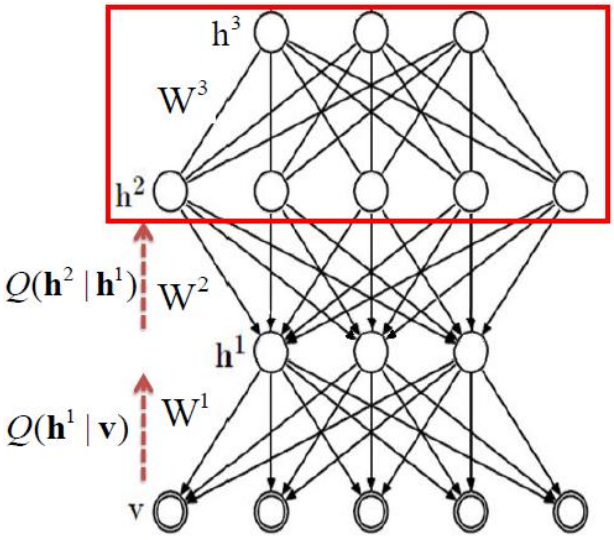


**Figure 2-19: Adding a new top hidden layer, freezing second RBM weights and training the third RBM in DBN structure**

In the proposed model in [10], for handwritten digits classification (over MNIST dataset [60]) both of two upper layers form an undirected associative memory (Figure 2-21). Other hidden layers construct a directed acyclic graph (as a generative model) to convert the representations in the associative memory into observable variables such as the pixels of an image. As was mentioned the learning algorithm is unsupervised but the model can learn a joint distribution of both label and data. This hybrid model, can uses any type of data as labels, but in [10] just the class labels for handwritten digits was used.

After training process the high level of abstraction can be generated in highest hidden layer of DBN. Generally DBN can be used in two kinds of applications. According to Figure 2-21 there is two (red and blue paths) paths which the network can be navigated. The first path (red path) including classification tasks and usually are followed with a *fine-tuning algorithm*. *Fine-tuning algorithm* is a supervised discriminative algorithm which can be used at the higher layer of DBN. This recent algorithms uses the extracted features produced by the DBN. In this category of applications the main role of Deep Belief Nets, is feature extraction. It was proven in [29] using this extracted features (as a pre-training phase) can improved the accuracy of classification algorithms. Also in [61-63] the advantage of pre-training in real applications was shown.

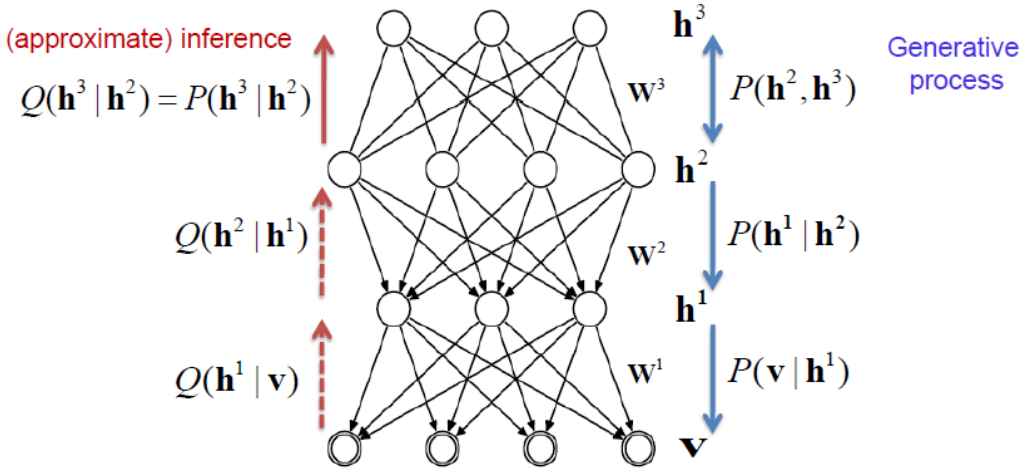


**Figure 2-20: Using Deep Belief Networks for approximation, classification and feature extraction (red path) and using as generative model (blue path)**

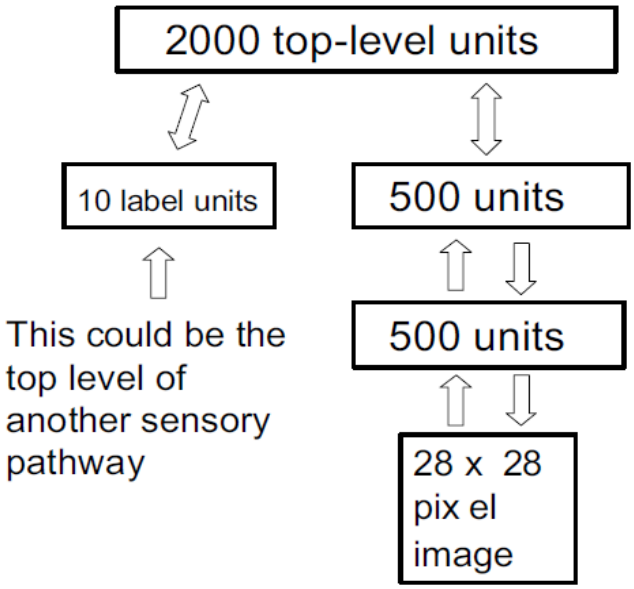


**Figure 2-21: The proposed architecture in [10] to learn both handwritten digits and corresponding labels using DBN.**

The second category consists of applications including generative models. In such cases network using a top-down algorithm, try to represent its *belief* about the input data (test vector). In the other words, this approach can help investigation of *remembering, imagination and concept representation process* in *mind*. The generative process, is similar to imagination in mind. For example when a label is presented to related units Figure 2-21, using a reverse process trained network can represent its inference as a vector of pixels.

### 2.4.4 DBN applications

Nowadays using deep architectures in state of the art technologies is very common and the advantage of deep networks was proven. Dealing with a huge amount of unlabeled data (*Big Data*), needs very efficient algorithm. Using Deep Belief Networks, which can be used in generative and discriminative tasks, is a big opportunity. As was mentioned, the output of a DBN can be used as the input for another model and give better results. In [64] DBN was used for processing and classification phone calls. The application is a phone call router which is responsible for routing each specific call to the desired path. In this application the network is pre-trained using a DBN. The extracted features (as the output of DBN) was applied to a *Multi-layer Feed Forward Network (FFN)* and classification was performed using *Back propagation* algorithm. The results are illustrated in Table 2-1. The results show the performance of the proposed model in contrast to some other machine learning algorithms.

Table 2-1: The results of [64] in using DBN

| Action Classification Accuracy (%) | | | | |
|---|---|---|---|---|
| Labeled Data | MaxEnt | SVM | Boosting | DBN |
| 1K | 76.0 | 77.8 | 79.6 | 78.1 |
| 2K | 80.4 | 82.2 | 83.6 | 82.6 |
| 3K | 82.2 | 84.3 | 85.1 | 84.4 |
| 4K | 83.5 | 85.3 | 84.6 | 85.5 |
| 5K | 84.6 | 86.2 | 85.9 | 86.2 |
| 6K | 85.5 | 87.0 | 86.3 | 87.0 |
| 7K | 86.2 | 87.7 | 86.3 | 87.8 |
| 8K | 86.5 | 88.0 | 87.2 | 88.0 |
| 9K | 87.2 | 88.5 | 87.5 | 88.7 |
| 10K | 87.6 | 88.5 | 87.7 | 88.7 |
| 27K | 89.7 | 90.3 | 88.1 | 90.3 |

Also the announced results in [10] demonstrate the high accuracy of using DBN in handwritten digits classification task (Table 2-2).

*Table 2-2: Comparison between DBN and some other methods in classification task on MNIST benchmark [10]*

| Learning algorithm | Test error % |
|---|---|
| Our generative model<br>784→500→500↔2000↔10 | 1.25 |
| Support Vector Machine<br>degree 9 polynomial kernel | 1.4 |
| Backprop 784→500→300→10<br>cross-entropy & weight-decay | 1.51 |
| Backprop 784→800→10<br>cross-entropy & early stopping | 1.53 |
| Backprop 784→500→150→10<br>squared error & on-line updates | 2.95 |
| Nearest Neighbor<br>All 60,000 examples & L3 norm | 2.8 |
| Nearest Neighbor<br>All 60,000 examples & L2 norm | 3.1 |
| Nearest Neighbor<br>20,000 examples & L3 norm | 4.0 |
| Nearest Neighbor<br>20,000 examples & L2 norm | 4.4 |

One of the best applications of Deep Belief nets was introduced in [30] in *reducing the dimensionality of data*. In [30] the results of dimension reduction is compare with *Principal Component Analysis (PCA)* and the advantage of proposed algorithm is proven. Using DBN in object recognition tasks [65, 66], speech and voice recognition [42-44, 63, 67, 68] and image and medical images processing [38, 39, 69-71] demonstrated very good results. Large research groups of top universities such as University of Toronto, Université de Montréal, Stanford University, University of Oxford and UC Berkeley and big companies such as Google, IBM and Facebook are working on deep learning [72]. The application of deep learning is growing up in various fields of technology.

# Chapter 3

## Spike Based Deep Belief Network

According to 2.2.1, the neural network issue, can be used considering two different point of views. As was mentioned in Artificial Neural Network, the mathematical aspect of neurons for computations and predictions is important. Despite, in Spiking Neural Network the biological aspect is serious. Indeed in SNN, spikes are determinative. A single spike does not contain any information, but the relation between a stream of spikes can transfer information. Spike timing and spike rate are two significant parameters in computation using SNN. This generation of neural networks because of using electrical circuit simulation, are suitable for VLSI implementation. As disused in previous chapters Deep Learning is a powerful bunch of algorithms, including Deep Belief Networks, for machine learning. The efficiency of these algorithms are demonstrated in some principal applications. Respecting to these two recent significant characteristics of SNN and Deep Learning from ANN, motivated us to propose a Deep Neural Network (which is powerful) of spiking neurons (which is suitable for VLSI implementation). But there are some challenges to overcome.

# 3 Spike Based Deep Belief Network: An online learning strategy for DBN with LIF neurons

To distinguish the problems in the progress of utilizing deep algorithms in Spiking Neural Networks, we must to be more familiar with SNN and some important related issues. In this chapter after introducing SNN types, we will have a look at learning in SNN and then we will discuss about online and offline learning approaches and finally we present our proposed method.

## 3.1 Spiking Neuron's Models

Similar to Artificial Neural Networks, some different model of neurons are proposed in spiking domain with specific properties and applications. In this section some of the most useable methods are explored.

### 3.1.1 Hodgkin & Huxley

Hodgkin & Huxley (HH) neuron model is the first conductance based model which is proposed by A. L. Hodgkin and A. Huxley in 1952. For this work they received the 1963

Nobel Prize in Physiology and Medicine. A. L. Hodgkin, A. Huxley had developed their model on propagation of action potentials in the squid axon [73-76]. The changes of membrane voltage, in real model, is because of the flow of Ions. The Hodgkin–Huxley model Ion channels and Ion current flow, can explain the spike generation process. In fact there are several types of channels in cell membrane which the corresponding ions can transfer through and lead to membrane potential difference between inside and outside of a cell. Sodium ($N_a^+$), Potassium ($K^+$), Calcium ($Ca2^+$) and Chloride ($Cl^-$ ) Ions can changes the potential difference. The Hodgkin–Huxley model, simulates the cell membrane using a capacitor ($C_m$) which its capacity is depended on voltage difference ($V_m$) and the current which is flowing ($I_c$) (Figure 3-1).

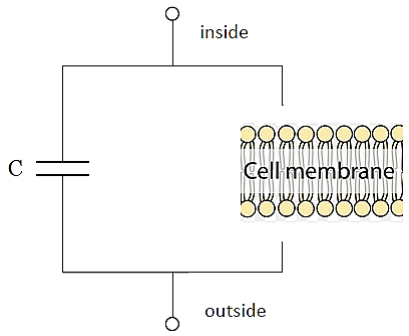


**Figure 3-1: From Hodgkin and Huxley experiences the membrane can be illustrated as a capacitor [74]**

The membrane voltage difference causes to membrane current. The dependency of membrane current to membrane voltage is explained in Equation (3-1).

$$I_m = C_m \frac{dV_m}{dt} \qquad (3\text{-}1)$$

Each types of Ions has its specific channels. Each channels has its current flows ($I_{ion}$). The channel's current is depended on the conductance of channel ($G_{channel}$), also each channel has a *revesal potential* (*Nernst potential*). Indeed we can assume the channels as some connected resistors in parallel with the membrane capacitor. Accordingly for any given type of Ions the current flow through the channels is explained in equation (3-2).

$$I_{Channel} = G_{Channel}(V_m - V_{Channel}) \qquad (3\text{-}2)$$

For the sack of simplicity in nerve membrane model, the basic model of Hodgkin-Huxley is based on the squid axon which has a simple system with basically only two types of voltage-dependent conductance, $G_{Na}$ and $G_K$. Since for each RC circuits we can assume a leaky current, then we can postulate another conductance for the leakage ($G_l$) and

corresponding leaky current ($I_l$) and also leaky reversal potential ($V_l$) (Figure 3-2).

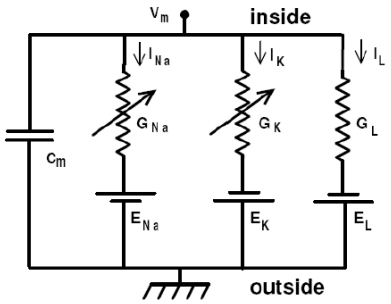


**Figure 3-2: Proposed electrical circuit by Hodgkin and Huxley for squid axon [74]**

The variable resistances in Figure 3-2 represent the conductance dependency to membrane voltage difference. Therefore for a simple cell of squid (only has sodium and potassium channels), equation (3-3) shows the total current through the membrane.

$$I = C_m \frac{dV_m}{dt} + G_{Na}(V_m - V_{Na}) + G_K(V_m - V_K) + G_L(V_m - V_L) \qquad (3\text{-}3)$$

The Hodgkin-Huxley model can shows the diversity of dynamical features of real neurons, but is highly non-linear because of the number of variables in equations. Then mathematical analysis can be more complicated for Hodgkin-Huxley model and leads to computationally expensive simulations of networks using Hodgkin-Huxley neurons. Therefore several more simple models are presented such as Integrate-and-Fire Model (I&F).

### 3.1.2 Integrate-and-Fire Model (I&F)

*Integrate-and-Fire* is a class of spiking neuron models. The simplest model in this class which can be described using a linear differential equation for computing the membrane voltage and defining a threshold for spike firing is the *Leaky Integrate-and-Fire* model.

Respecting to limitation of the “leaky integrate-and-fire” model in several respects, some Generalized Integrate-and-Fire neurons was proposed.

Generalized integrate-and-fire models consisted of some basic model. The first one is *Nonlinear Integrate-and-Fire* neurons which the *Quadratic* and *Exponential Integrate-and-*Fire model are special case of this model. Using an adaptive threshold, the predictions of a simple Integrate-and-Fire model can be improved. *Adaptive Integrate-and-Fire models* are the second generalized model of simple Integrate-and-Fire model including the *Izhikevich*

model and *Adaptive Exponential Integrate-and-Fire* model. In this dissertation, because of using LIF model, we don't discuss more about the other types of I&F models.

### 3.1.2.1 Leaky Integrate-and-Fire Model (LIF)

The basic model of integrate-and-fire neurons, was proposed in 1907 by Lapicque. Indeed it's before understanding the mechanisms of action potentials generation, but despite the simplicity of integrate-and-fire model, it is still a useful description of neuronal activity [77].

The simplest form of an integrate-and-fire model is consisted of a capacitor and a resistor, which are coupled in parallel. The input current can be generated by an external current source or via synaptic input from presynaptic neurons. Considering Figure 3-3, the driving current leads us to equation (3-4):

$$I(t) = I_R + I_C \tag{3-4}$$

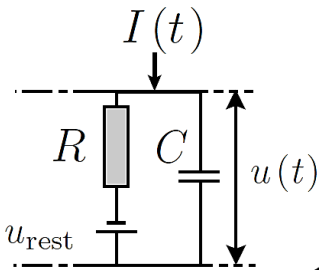


**Figure 3-3*: Basic model of Integrate-and-Fire neuron***

Respecting to $I_R = u_R/R$ ($u_R = u - u_{rest}$) and $C = q/c$, where q is the charge and u is the voltage, we have $I_C = C\frac{du}{dt}$, we have equation (3-5):

$$I(t) = \frac{u(t) - u_{rest}}{R} + C\frac{du}{dt} \tag{3-5}$$

Assuming $\tau_m = RC$, we have the standard form of integrated-and-fire model equation (3-6):

$$\tau_m \frac{du}{dt} = -[u(t) - u_{rest}] + RI(t) \tag{3-6}$$

Where *u* can explain the membrane potential. In some literatures (3-6) is called a *passive membrane*.

The driven differential equation for our RC-circuit (3-6), with respecting to $u(t_0) = u_{rest} + \Delta u$ and $I(t) = 0\ for\ t > 0$, can be solved (3-7) [26].

$$u(t) - u_{rest} = \Delta u \exp\left(-\frac{t - t_0}{\tau_m}\right) \qquad (3\text{-}7)$$
$$where \quad \tau_m = RC \,, \mathrm{t} > \mathrm{t}_0$$

As we can see, when the input is removed after t=0, the membrane potential decreases exponentially to $u_{rest}$. This solution is called free solution (No input for t>0).

If equation ( 3-6), injected by a constant input I(t)=I0 for t between 0 and Δ, and supposing $u(0) = u_{rest}$ for simplicity. Integrating (3-6) with the chosen initial conditions, we have (3-8):

$$u(t) = u_{rest} + RI_0\left[1 - \exp\left(-\frac{t}{\tau_m}\right)\right] \qquad (3\text{-}8)$$

For constant $I_0$, when $t \to \infty$ we have $u(\infty) = u_{rest} + RI_0$, and it is the maximum amplitude of membrane voltage with constant input current. Indeed it is the *steady state* of RC circuit. As depicted in Figure 3-4, for some given parameters R=0.1, C=0.1 and $I_0$=0.1during 1000 ms, the maximum value for *u* is $RI_0$=*0.01Volt*.

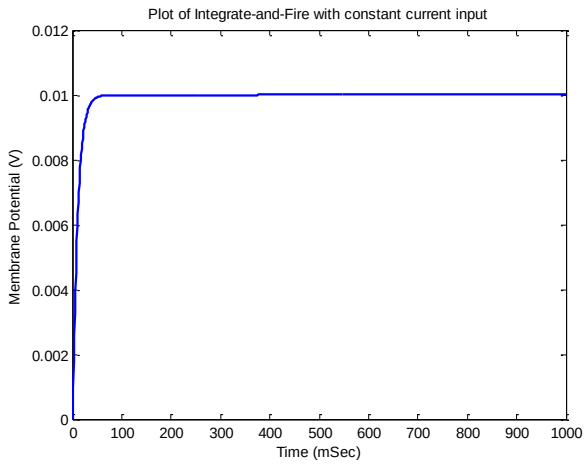


**Figure 3-4*: Membrane potential for constant input current***

Using inputs with short period of time (pulse), the maximum value cannot be reached. In Figure 3-5 we can see the effect of pulse time and $\tau_m$. If the pulse duration is longer than $\tau_m = RC$, the circuit can reaches *steady state* (Figure 3-5-a), but with pulse duration less than $\tau_m$, the maximum value is not reachable (Figure 3-5-b).

For short time pulse, we can use mathematical abstraction of $\delta(t)$ which is called the Dirac δ –function ($\delta(t)$ duration is $(\Delta \mathrm{t} = t_2 - t_1) \ll \tau_m$).

For a more realistic model, we have to consider a network of neurons, the input current for any neurons is driven by presynaptic neurons regarding to the strength of synaptic

efficacy between pre- and post-synaptic neurons. The total currents in a given post-synaptic neuron i, from all pre-synaptic neurons j, can be expressed as (3-9) [28]:

$$I_i(t) = \sum_j W_{ij} \sum_f \alpha(t - t_j^f) \qquad (3\text{-}9)$$

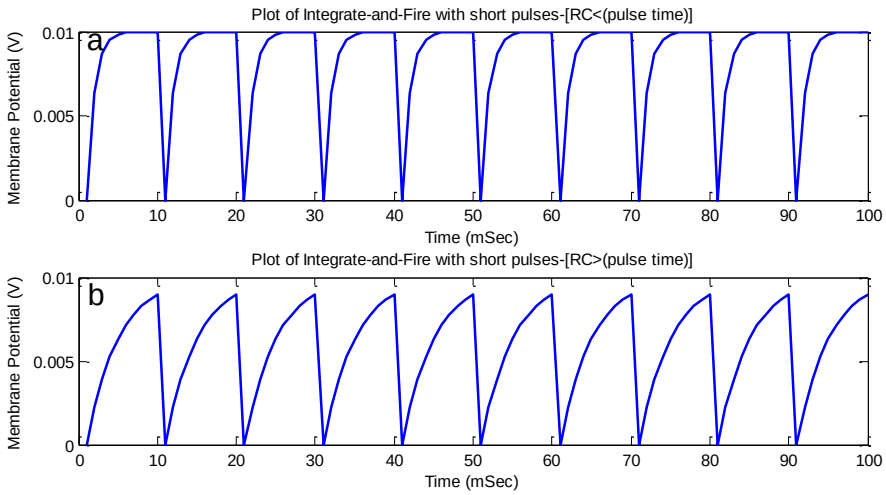


**Figure 3-5: The effect of pulse duration in contrast to $\tau_m$ for some iterated pulses.**

Where $W_{ij}$ is the efficacy of the synapse between neuron j to neuron i, and $t_j^f$ is the firing time of neuron j. $\alpha - function$ is used to define the time course of input current in neuron i, when neuron j spiked at $t_j^f$ (more details are described in [26, 28]).

The leaky integrate-and-fire model as the simplified neuron model cannot explain many features of real biological neurons, but this model can precisely imitate the spike generation of neurons in proper time of events. Indeed, integrate-and fire model is a perfect model of spike generation in neurons. Respecting to this feature and the simplicity of the model (less computational is needed), we use this model for our implementation.

### 3.1.3 Izhikevich Model

There are some evidences [78] which express the response of neurons is consisted of some different patterns of spike generation with corresponding inter-spike intervals known as firing pattern.

Figure 3-6 shows the variety in firing patterns. The Adaptive Exponential integrate-and-fire (AdEx) is an extension of integrate-and-fire model exploiting adaptation, can describe firing patterns [26].

Another model which using quadratic integrate-and-fire can explain the firing patterns, is Izhikevich model. Izhikevich is an abstract neuron model combining the biological dynamics of Hodgkin–Huxley-type and the computational efficiency of integrate-and-fire neurons. In [78] professor Izhikevich shows how using this model, we can simulate a large spiking network using many low computer resources.

The proposed model by adjusting four parameters (a,b,c,d), can regenerate all the know firing patterns of cortical neurons.

The model is represented in (3-10):

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \qquad (3\text{-}10)$$
$$\frac{du}{dt} = a(bv - u)$$

Respecting to (3-11):

$$if\ v \geq 30mV, then \begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \qquad (3\text{-}11)$$

In [79] professor Eugene M. Izhikevich, discussed about the biological aspect and computational efficiency of some of the most useful models and compared the models in both aspects of biological plausibility and computational efficiency. Using these comparisons, researchers can select the suitable model depending on the spike dynamics which can satisfy their requirements. As was mentioned in [79] in simulation process if the spiking models don't choice inappropriately, may leads to incompatible results. In Figure 3-8 and Figure 3-9 we can see the comparison results. The cost of implementation (FLOPS) is approximation of required number of floating point operations in 1ms simulation.

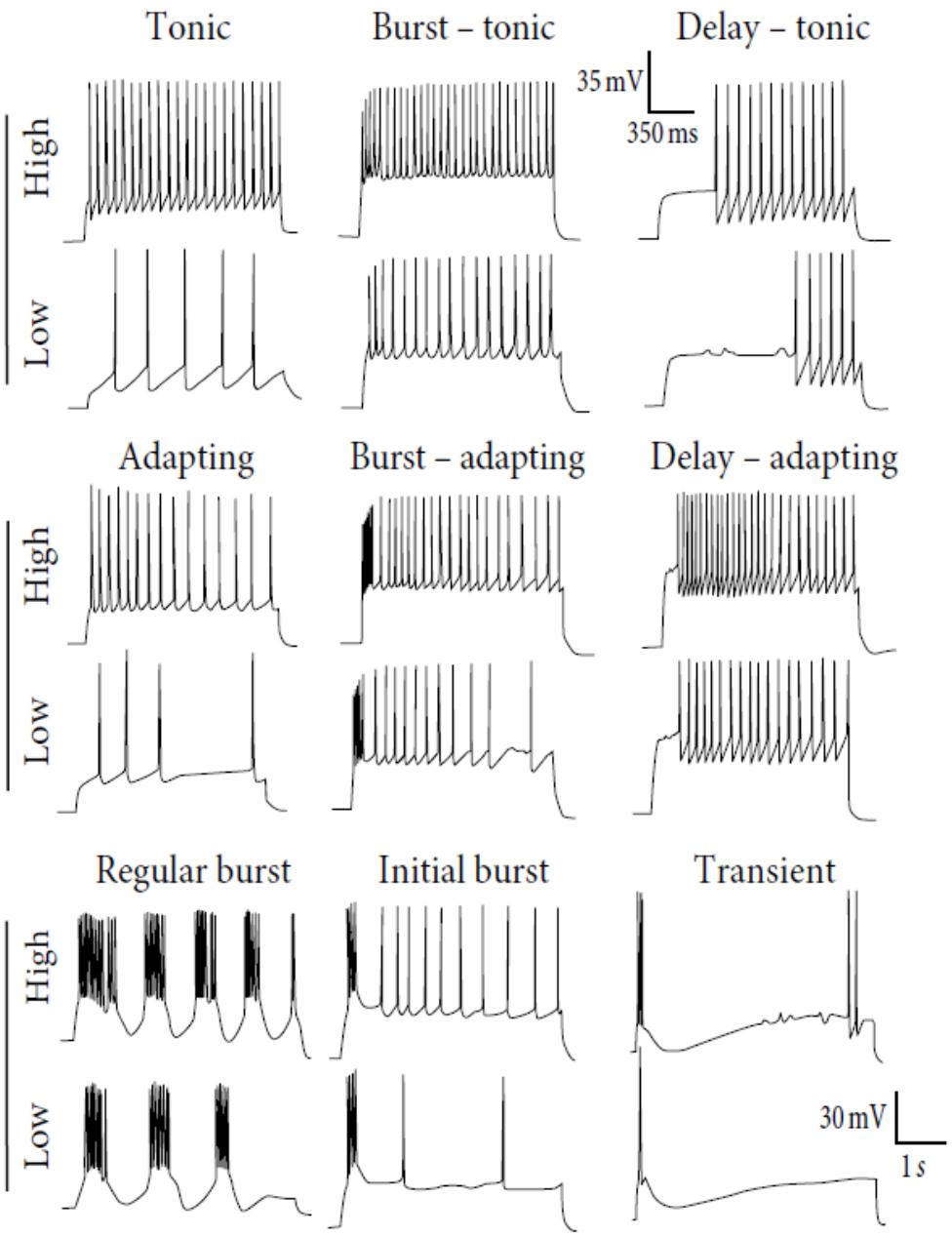


*Figure 3-6*: Multiple firing patterns in cortical neurons

**Different types of firing patterns can be generated by adjusting the model parameters (Figure 3-7). According to firing patterns neurons in cortical are classified into some types, which neurons in each types are excitatory or inhibitory. All excitatory and inhibitory type parameters can be found in [78].**

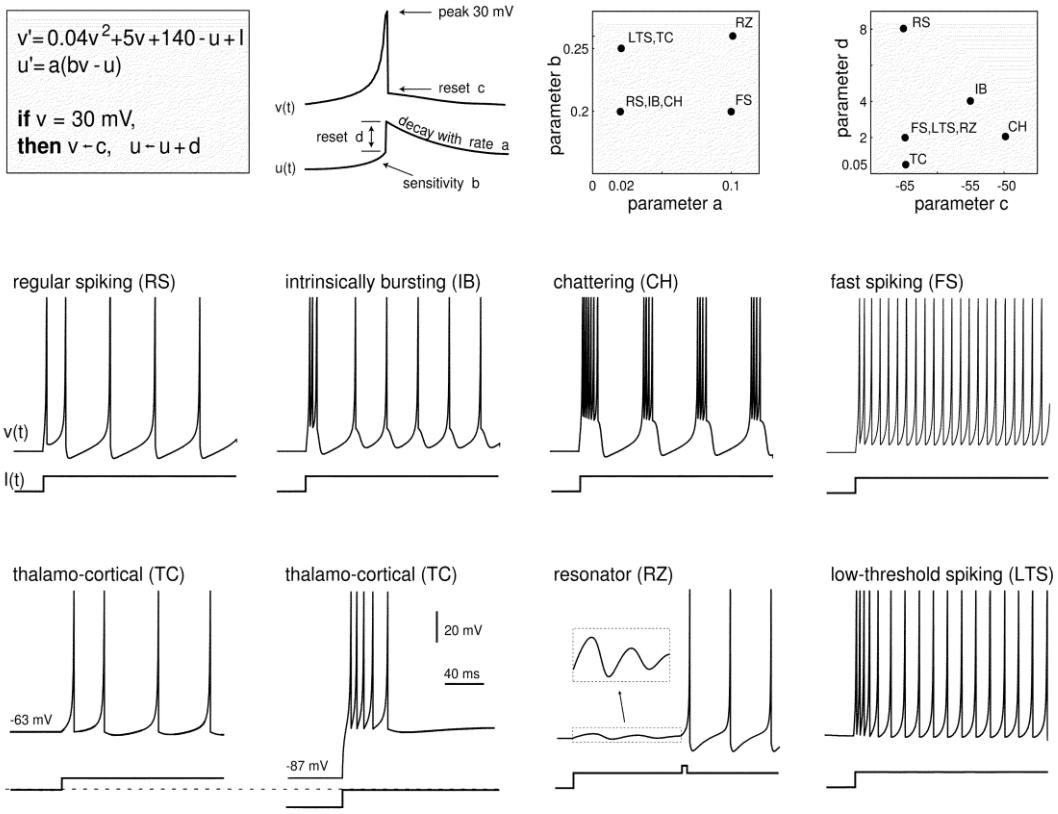


**Figure 3-7: Adjusting parameters a,b,c and d known different types of dynamics can be reproduced**

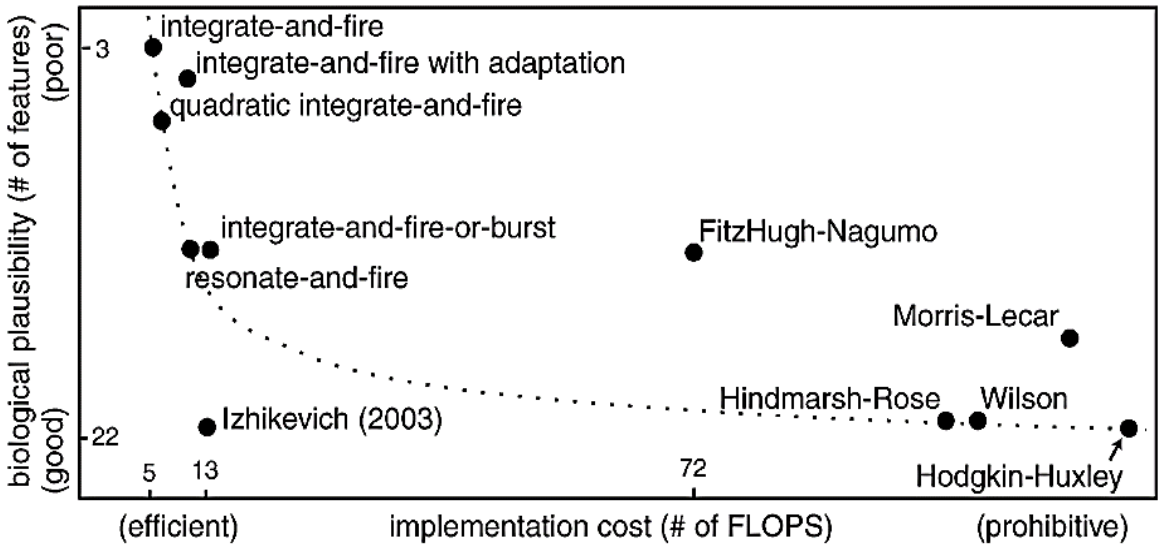


**Figure 3-8*: Comparison of the properties of spiking models (1)***

The mathematical analysis of the model are published in [80] also professor Izhikevich in his website [81] has uploaded some perfect Matlab codes for implementing all dynamic types. In addition professor Izhikevich and their colleagues are working on a project called "Brain Corporation" [82] which is intended to exploiting machine learning and computer vision to create intelligent systems.

| Models | biophysically meaningful | tonic spiking | phasic spiking | tonic bursting | phasic bursting | mixed mode | spike frequency adaptation | class 1 excitable | class 2 excitable | spike latency | subthreshold oscillations | resonator | integrator | rebound spike | rebound burst | threshold variability | bistability | DAP | accommodation | inhibition-indiced spiking | inhibition-induced bursting | chaos | # of FLOPS |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| integrate-and-fire | - | + | - | - | - | - | - | + | - | - | - | - | + | - | - | - | - | - | - | - | - | - | 5 |
| integrate-and-fire with adapt. | - | + | - | - | - | - | + | + | - | - | - | - | + | - | - | - | - | + | - | - | - | - | 10 |
| integrate-and-fire-or-burst | - | + | + | | + | - | + | + | - | - | - | - | + | + | + | - | + | + | - | - | - | | 13 |
| resonate-and-fire | - | + | + | - | - | - | - | + | + | - | + | + | + | + | - | - | + | + | + | - | - | + | 10 |
| quadratic integrate-and-fire | - | + | - | - | - | - | - | + | - | + | - | - | + | - | - | + | + | - | - | - | - | - | 7 |
| Izhikevich (2003) | - | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | + | 13 |
| FitzHugh-Nagumo | - | + | + | - | | - | - | + | - | + | + | + | - | + | - | + | + | - | + | + | - | - | 72 |
| Hindmarsh-Rose | - | + | + | + | | | + | + | + | + | + | + | + | + | + | + | + | + | + | + | | + | 120 |
| Morris-Lecar | + | + | + | - | | - | - | + | + | + | + | + | + | + | | + | + | - | + | + | - | - | 600 |
| Wilson | - | + | + | + | | | + | + | + | + | + | + | + | + | + | + | | + | + | | | | 180 |
| Hodgkin-Huxley | + | + | + | + | | | + | + | + | + | + | + | + | + | + | + | + | + | + | + | | + | 1200 |

**Figure 3-9*: Comparison of the properties of spiking models (2)***

## 3.2 Learning in Spiking Networks: Spike-based vs. Rate-based learning

Dependency of postsynaptic responses to the arrived action potentials, in neural network models, is shown using $w_{ij}$ as the synaptic weight. The strength of this dependency can be changed during *learning process*. In theory of neural networks the weight $w_{ij}$ of a connection between two neurons can be adjusted to form a specific task.

Indeed *learning* is the process of parameters adjustment to adapt the parameters to a given task. These parameter adjustment procedure define a *learning rule*. In biological neural networks, in some situations, synaptic changes can be caused by considering the *correlation* between pre- and post-synaptic neurons activity. This type of *learning rule* is called *Hebbian learning*.

Many models of biological learning rules are inspired by Hebb's rule [83] that describes the modification process of connection between presynaptic neuron A and postsynaptic neuron B:

"When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both

cells such that A's efficiency, as one of the cells firing B, is increased."

The Hebbian rule has been the basis of learning methods in Spiking Neural Networks. The pre- and post-synaptic correlations for adjusting the synaptic parameters, was considered in two aspects of *rate of firing* and *time of firing*. As was mentioned in order to change the synaptic connections strength, both of pre- and postsynaptic neurons must to be activated. This activation of pre- and post-synaptic neurons can be considered as correlation between pre- and post-synaptic *firing rate*. Also this correlation can be affected by the correlation between the *spike-time*. These two aspects of correlation, are originated from the mentioned models: *rate-based* and *spike-based* model which lead to two different *learning rules: rate learning and temporal learning.* In the following we discus rate-based Hebbian learning as a rate based learning rule and STDP as a temporal learning approach. But this is considerable whether brain use rate based learning or temporal based learning. Both aspects of Hebbian learning from neural coding have been formulated and compared in [27, 84-88].

In [89], author discus about both of rate-based and spike-based theories and presents some significant question such as *"Does the brain use a rate code or a spike timing code?"* and shows such questions are need to be rephrased.

In fact these points of views, are two types of models of the brain: spike-based models and rate-based models. These models have been formed base on the observations. From the spike-based perspective, spike trains are observable. For example in a given spiking neuron model the spikes can change the neuron's membrane potential which leads to spike generation or nothing. In this type of model the observable parameter is the neuron's state. In the other words, in spike-based model the relation between input spikes train and the output one is considerable. The observable parameters in the rate-based model, are rates. Rate can be defined in various forms of average over spikes. In this regard, rate-based model defines rates relationships over time.

But as we know the rate measurement is not observable independently. Indeed spikes which are generated through interactions between pre- and post-synaptic neurons, define the rates. Generally dynamics of spiked-based models can be approximated using rate-based models. Also [89] talks about some assumptions for reducing a spiked-based model to a rate-based model which means inference in the rate-based instead of inference in the spike-based model .

Consequently both models, are two aspects of one phenomena and express the relation between *sensory stimulus* and *neural activities* and they describe the way neurons interact with each other [89].

In the following we talk about *Neural Coding*: Spike-based and rate-based models which encode spikes using *rate coding* and *spike coding*.

### 3.2.1 Neural Coding and Data Processing in Brain

The brain receives too many information from the world through the sensory system. This sensory inputs will be converted to streams of action potentials. The action potentials can propagate in the neural network and transfer the information from world to brain to decide for appropriate action. These external sensory stimulis must be encoded to carry information. The *neuronal coding* and signal transferring is still unsolved questions, but the relevance of temporal information and rate of firing for some neuronal systems has been clearly shown [28]. One of the fundamental problems in neuroscience, is neuronal coding. The general thought about neuronal coding is on the *mean firing rate* of the neuron as the most popular method for signal transferring.

As was mentioned in previous section, spike timing and rate of spikes are two interpretation of neural dynamics. Consequently, defining spike trains using firing rates or using temporal aspects of spikes cannot define the neural coding for sensory stimulis and some spike coding schemes cannot be defined. Indeed in neurons used various type of neural coding for information transferring. Firing rate and temporal codes are main point of view in neural coding. In the following we discuss these recent aspects of neural coding. But since our proposed model is based on rate aspect of neural coding, we only explain the rate based coding schemes. Then we explain *Rate-Based Hebbian Learning* and *Spike Time Dependent Plasticity*.

#### 3.2.1.1 Rate Codes

Rate codes as a traditional scheme of coding, explain how with using firing rate concept, neurons can code the input signals can transfer information. In fact this theory is based on the stochastic property of neural firing patterns. Experimentalists show that neural activities for a given experiment are not same from trial to trial. Then we can deal with this non-stationary and stochasticity, by using of probability and statistics. The averaging is a statistical method which can explain some expected value of a phenomena.

A general definition of firing rate (regarding population activity), is instantaneous average of generated spikes of neurons then firing rate can be defined as temporal average. The number of spikes in a given time window (typically 100ms or 500ms) is the most usual definition for mean firing rate. But as was explained in [26] there are at least three different definition of rate which are discussed in this section.

#### 3.2.1.1.1 Rate as a Spike Count (Average over Time)

*Temporal average*, as the number of spikes in an time interval (*T*), is the most common definition of rate.

According to experimentalist's researches, the practical time window to record sensible averages over several spikes are *T = 100ms* or *T = 500ms*, but the it is depended on the type of neuron.

As depicted in Figure 3-10the temporal average can be defined as (3-12) where $N_{sp}$ is the number of captured spikes in the time interval (*T*).

$$Temporal\ Average = \frac{N_{sp}}{T} \qquad (3\text{-}12)$$

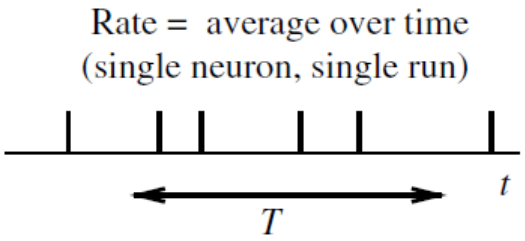


**Figure 3-10: Rate can be defined as the number of spikes in a given time (T)**

In [28] several successful practical work using this definition of mean firing rate, are mentioned. Also mentioned some classical results which show that the spike count measure, can evaluate and classify the neuronal firing. But we must notice about the reaction time. In reality, the reaction time is usually rather short than the proposed time interval for averaging. This time maybe short as 30-40ms. This time interval is not long enough for averaging using the number of spikes.

Since the inputs are not stationary, then the temporal averaging can be used successfully when the stimulus are constant during the experiment or the changes are occurred slowly. But based on the theory that "a neuron transforms information about a single input variable (the stimulus strength) into a single continuous output variable (the

firing rate)" this aspect of a neural code is used in too many models of neural networks.

In temporal averaging when the stimulus strength increased the value of output rate increases consequently. Continuing this process the output will saturate (will reach a maximum value).

Because of using current as stimuli, the frequency-current curve can shows the relation between the measured firing rate (frequency) and the input stimuli (current).

Respecting to noise and irregularities in spikes trains in the cortex, this aspect of rate coding is a practical way to average over a larger number of spikes.

#### 3.2.1.1.2 Rate as a Spike Density (Average over Several Runs)

Another interpretation of mean firing rate is presented based on running the experiment for several times. In each trial we can capture the results when the neurons are stimulated with inputs. In each trail we assume same situations.

After several trials the neuron response can be shown using a Peri-Stimulus-Time Histogram (Figure 3-11).

In Wikipedia PSTH is defined as following:

"In neurophysiology, peristimulus time histogram and poststimulus time histogram, both abbreviated PSTH or PST histogram, are histograms of the times at which neurons fire. These histograms are used to visualize the rate and timing of neuronal spike discharges in relation to an external stimulus or event."

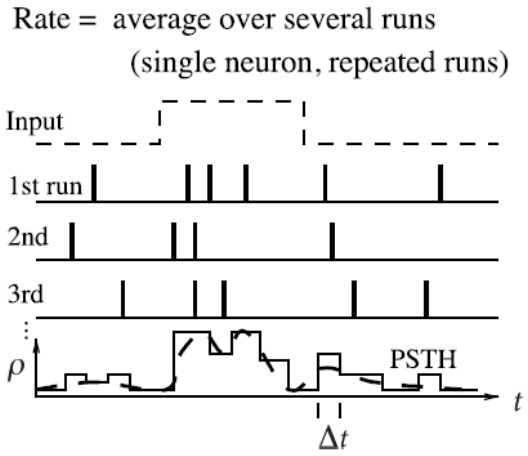


**Figure 3-11: PSTH is a histogram from neural activities which can present rate and time of neuronal spike**

In PSTH the results can be smoothed to have a continuous value for rate parameter. The spike density as a time depended firing rate often presented in units of Hz.

Considering the time-dependency of the spike density, this model is a useful measure to evaluate neuronal dynamics, but the problem with this method is that in real models the experiment may run just one time.

#### 3.2.1.1.3 Rate as a Population Activity (Average over Several Neurons)

Because of the problem of the spike density model with number of trials, we can assume there are large populations of independent neurons and they are triggered using the same input stimulus. Then instead of capturing neuron activities of a single neuron in several runs, the dynamics of many neurons in a single run will be recorded.

In fact in visual cortex the mentioned conditions can be found ([90]). The number of neurons in the brain which are involved in a process is huge and many of them usually have same input-output characteristics. Then when a population of *m* neurons are triggered with same input (Figure 3-12), then the neuron will activated with same pattern of activity. The triggered population can sent their generated spikes to another population with *n* neurons. In an ideal situation all the pre-synaptic neurons in first population are connected to the second one (Figure 3-13).

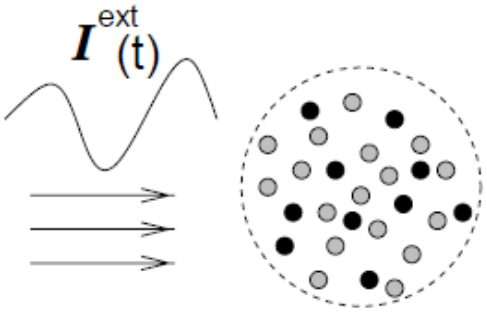


**Figure 3-12: All the neurons in same population receive same inputs [91]**

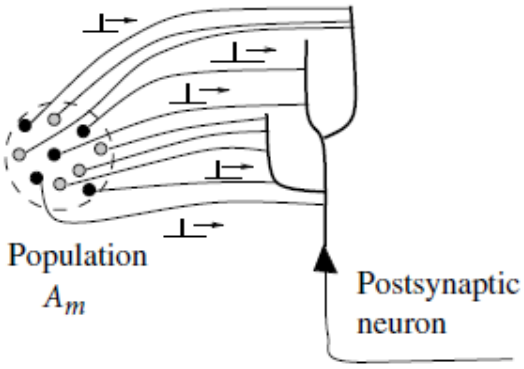


**Figure 3-13: The pre-synaptic population's neurons are connected to pots-synaptic population**

Then the activity measurement $A_m(t)$ for a given population $m$ is defined in (3-13) when $\Delta t$ is small time window and $n_{act}(t, t+\Delta t)$ is the number of spikes in the given time $\Delta t$ and $N$ is the number of neurons in the population $m$ (Figure 3-14).

$$A_m(t) = \frac{1}{\Delta t}\frac{n_{act}(t; t+\Delta t)}{N} \qquad (3\text{-}13)$$

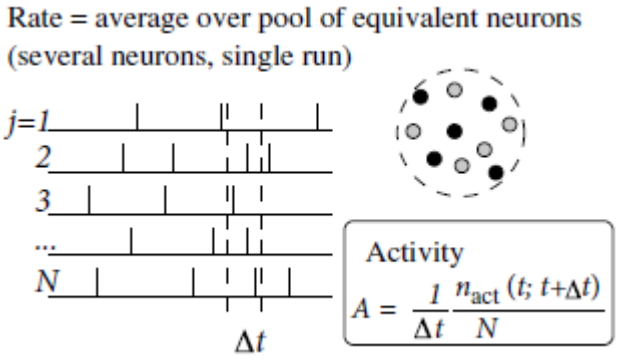


**Figure 3-14: Rate definition as average over a population of neurons with same neural properties**

### 3.2.1.2 Spike Codes:

Since the rate coding approaches are probabilistic then cannot present precise features of neural activities. Therefore several schemes which are based on spikes are proposed in [26, 28] as bellow, but respecting to our proposed method for implementation, we are not focused on rate aspect of neural coding and here for the sake of completeness we only list the title of spike coding approaches:

- Time-to-First-Spike
- Phase
- Correlations and Synchrony
- Stimulus Reconstruction and Reverse Correlation

## 3.2.2 Rate-Based Hebbian Learning

Since Hebbian rule did not formulate in mathematical terms, originally, in [27, 28, 88] first classical formulations of Hebbian learning based on firing rates has been proposed.

In this rate-based model between the membrane voltage potential of a given neuron i ($u_i$),and its corresponding firing rate ($v_i$), a simple relation function g can be assumed such as (3-14):

$$v_i = g(u_i) \qquad (3\text{-}14)$$

The membrane potential $u_i$ as an average over time, can be measured as the total effect of pre-synaptic neurons activities. Assuming $v_i$ as the firing rate of each given pre-synaptic neuron j, we have the total effect of pre-synaptic neurons in a given post-synaptic neuron as

$$u_i = \Sigma_j w_{ij} v_j \qquad (3\text{-}15)$$

the integration of the receiving effects respecting to each link efficacy (3-15).

As it is obvious, the main parameter in $u_i$ and subsequently in $v_i$, is $w_{ij}$ which can be changed during learning process. Regarding this point of view, here the focus is on the rate of weights changes during one trial which the instantaneous change for a given synaptic connection can be shown as $\frac{dw_{ij}}{dt}$.

In Hebbian learning rule, when both of pre- and post-synaptic neurons are active, in a given trial, the strength of their connection will be increased. But, in the original Hebb model, no strategy for weight decreasing has been presented. To have a more mathematical model of Hebbian rule, which can describe the weight changes as a function of pre- and post-synaptic activities and to have a model of weight plasticity, respecting to Hebbian rule, in [27, 88] six important aspect of Hebbian rule have been considered.

The Hebbian learning rule was mathematically formulated focusing on a single pre-synaptic neuron which transmits signals using single synapse with efficacy $w_{ij}$ to a post-synaptic neuron. Respecting to each postulated aspect, some parameters have been added to the model. The first aspect is *Locality*. Locality implies that *"the synaptic changes are only depend on local variables"* [28]. Then the weight changes can be formulated as function of the local parameters in a connection between neuron i and j (3-16).

$$\frac{dw_{ij}}{dt} = F(w_{ij}; v_i, v_j) \qquad (3\text{-}16)$$

Where *F* function is not clear at this point. *Cooperatively* is another aspect. Indeed for a change in synaptic weight, pre- and post-synaptic neurons have to be activated simultaneously.

To implement *F* function with the desired properties, *F* has been extended (Taylor series) about - $v_i = v_j = 0$ [28, 88] as below:

$$\frac{dw_{ij}}{dt} \approx c_2^{corr}(w_{ij})v_i v_j + c_2^{post}(w_{ij})v_i^2 + c_2^{pre}(w_{ij})v_j^2 + c_1^{pre}(w_{ij})v_j + c_1^{post}(w_{ij})v_i + c_0(w_{ij}) + \mathcal{O}(v^3) \quad (3\text{-}17)$$

Respecting to this formulation the simplest form of Hebbian rule, as a correlation sensitive method can be expresses as:

$$\frac{dw_{ij}}{dt} = c_2^{corr}(w_{ij})v_i v_j \qquad where\ (c_2^{corr} > 0) \quad (3\text{-}18)$$

Equation (3-18), same as Hebb rule, dose not implies any weight decay approach. In network of neurons if the synaptic weights can only be strengthened, finally all the connections will get the maximum value of weights. In the original Hebb's no rule for decreasing the synaptic weights has been proposed. Therefore a rule for decreasing the weights (*synaptic depression*) is necessary for any useful learning rule. By defining *Synaptic Depression* aspect as $c_0(w_{ij}) = -\gamma_0 w_{ij}$ when $\gamma_0 > 0$, synaptic decay can be implemented.

*Competition* is another interesting aspect of learning rules. Considering this aspect, if a subgroup of synapses is strengthened, other synapses which are connected to the same postsynaptic neuron, have to be weakened. Some other aspects are *Boundedness*, and *Long-term stability*. Each of the aspects are formulated as a part of (3-25) in [28, 88].

### 3.2.3 Spike Time Dependent Plasticity (STDP)

Because of the averaging feature of rate coding, it is possible to neglect some time depended information. According to evidences form real biological systems, the synaptic effects can change during time. In neuroscience, changes of the synaptic strength are called *synaptic plasticity* [26]. In [85, 86] Richard Kempter et al., proposed a formulated version of correlation-based learning respecting to spike aspect of neural coding. Pre-synaptic and post-synaptic spikes dependency and its influences on synaptic weights during a "learning window" was investigated. This spike-based learning rule in a time-averaged can be interpreted as anti-Hebbian and Hebbian correlations between input and output spikes [87].

Focusing on temporal aspect of correlations between the spikes of pre- and post-synaptic neurons, leads to a temporally asymmetric form of Hebbian learning called *Spike Time Dependent Plasticity (STDP)*. The synaptic changes can be generated by synaptic plasticity. Spike-based learning considers temporal correlations between neuron's spike times.

In biological models new experiments can change the postsynaptic response. If the experiment cause to a persistent increase of the synaptic weight, the effect is called *Long Term Potentiation of synapses (LTP)*. If after the process the synaptic efficacy decreased persistently, a *Long Term Depression (LTD)* has been occurred. Spike Time Dependent Plasticity can explains LTP and LTD. In STDP if presynaptic spikes are arrived a few milliseconds before action potential of postsynaptic (in a short time window) frequently, long-term potentiation (LTP) of the synapses can performed. Inversely, arrival of spikes after postsynaptic spikes caused to long-term depression (LTD) of the synapses. This form of plasticity in synapses leads to learning and memory in the brain [92]. The synaptic changes can be demonstrated as function of the time interval between pre- and post-synaptic action potentials (Figure 3-15).

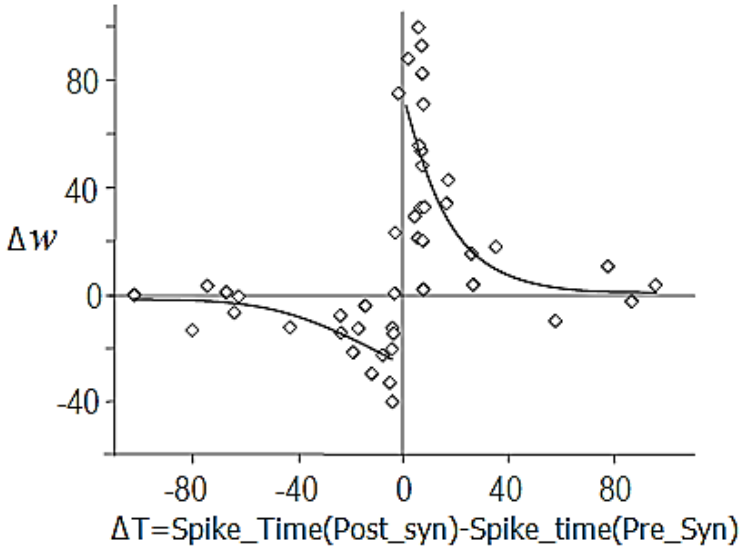


**Figure 3-15: The synaptic weights changed as a function of spike time of post and pre synaptic [92]**

Figure 3-15 show the STDP function or learning window. For $\Delta T < 0$ *Depression* of synaptic efficacy (LTD) occurs and when $\Delta T > 0$ *Potentiation* (*LTP*) will occur. The smaller timing window leads to more changes in synaptic strength. Indeed STDP can be defined using $W(t_{pre}\text{-}t_{post})$ function which can determines the synaptic strength according on the pre- and post-action potential times. The typical form for *W* function is shown in equation (3-19).

$$W(t_{pre} - t_{post}) = \begin{cases} A_+ \exp\left(-\dfrac{(t_{pre} - t_{post})}{\tau_+}\right) & if\ t_{pre} < t_{post} \\ A_- \exp\left(-\dfrac{(t_{pre} - t_{post})}{\tau_-}\right) & if\ t_{pre} > t_{post} \end{cases} \qquad (3\text{-}19)$$

Where the parameters $\tau_+$ and $\tau_+$ determine the temporal ranges and A- and A+ determine size of the changes.

## 3.3 Online vs. offline learning in hardware implementations

Considering our main goal, hardware accelerator for neural computations, we are interested in hardware implementations of Spiking Neural Networks. In practical applications there are two main approaches of designing (and training) Spiking Neural Networks for machine learning applications. One is training an Artificial Neural Network using corresponding algorithms and transferring the trained weights to an SNN [93-96], and another is learning the weights of the Spiking Neural Network using a biological inspired method such as Rate-based Hebbian learning or STDP [97-102].

Learning in first category of implementations is done in an *offline* manner. This implies when we use the implemented hardware, in ASIC or FPGA, all things are fixed and no parameters adjustment can be performed [94, 96, 103]. This type of application of Spiking Neural Networks cannot be used in other applications and only can satisfy a specific task. Indeed we need two framework for these category, one for training and another for testing and real application. In contrast to this *offline* method there is an *online* implementations which use same hardware for learning and main application. The recent category can be trained in future for new task (with some considerations). The *online* learning can done exploiting specific VLSI implementation of Spiking Neurons and Synapses. Some applications used CMOS transistor [14, 104] in VLSI scale. But since 2008, technology provides a unique and new opportunity for realization of *online* learning in Neuromorphic. Founding the missing *Memristor* [13], provides wide verity of Neuromorphic applications.

### 3.3.1 Memristor: A big opportunity for Neuromorphic

In 1971 professor Leon Chua, based on the symmetry of the equations for the resistor, capacitor and inductor, predicted the existence of the fourth device that govern a relationship between magnetic flux and charge as $d\varphi = Mdq$ (Figure 3-16). They are asymmetric two-terminal devices in which their polarity should be considered. Memritors behave as variable resistances in which the resistance can be controlled by voltage (flux) or through current (charge) and they can remember their last state (resistance).

In 2008, Hewlett-Packard (HP) Labs fabricated the first Memristor device according to the classical model (Generally two structures of Memristors have been fabricated. The

one which fabricated by the group at the Michigan University and the HP group model). Memristors provide high potential of applications development in wide variety. One of the very powerful aspects of Memristor it's their capability in keeping their last state in absence of electrical power. Indeed it could be a suitable non-volatile memories [105].

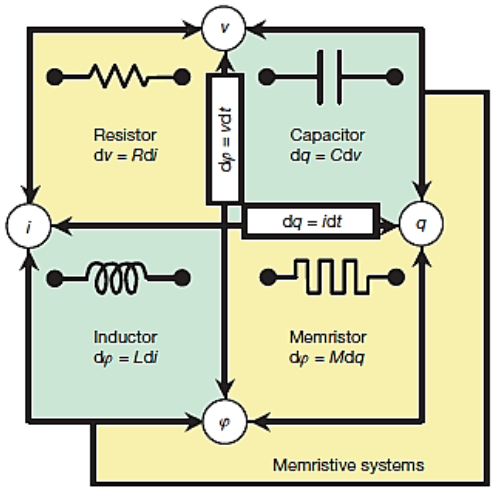


**Figure 3-16: Fundamental two-terminal circuit elements and the missing element:Memristor [13]**

The total resistance of the Memristor is determined as Figure 3-17 by two coupled variable resistors.

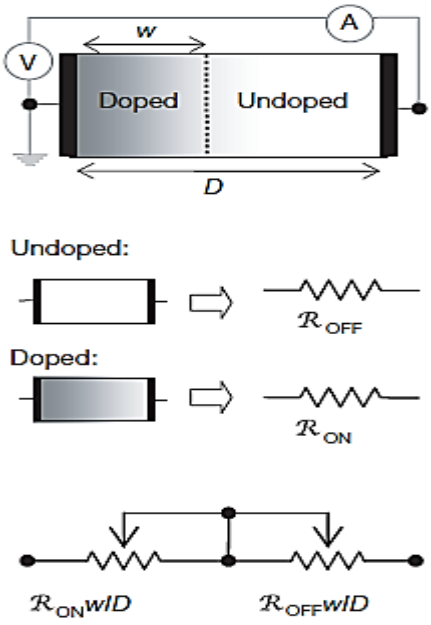


**Figure 3-17: Coupled variable-resistor model for a Memristor [13]**

## 3.4 Using machine learning algorithms in Spiking Neural Networks

The efficiency of machine learning algorithms for Artificial Neural Networks has been shown and there are too many work that can demonstrate the capability of machine learning algorithms in classification, clustering and pattern recognition. On the other hand, the possibility to have a specific hardware framework through using the electrical circuits of spiking neurons and exploiting VLSI systems, leads to have very dense chips of biological-like neurons in a very low power platform [5, 14, 104].

Both of them (the machine learning high accuracy and advantages of VLSI implementation for spiking neuron models), motivate researchers to have both of this ability in a same framework. As was expressed in introduction of this dissertation, and respecting to the concept of action potential as the main element in transferring signals between neurons of a network, it is obvious that there is big gap between Artificial Neurons and Spiking Neurons. Indeed the main problem in using machine learning method in spiking architecture, is in the nature of spikes. Spikes usually have same shapes and their ability to coding signals is depended on spike's timing and rate of spiks. But as we know timing has no matter in machine learning algorithms.

To overcome this problem some investigations have been done. To use spiking neurons in machine learning domain, we have to use *abstract models*. In [106] Neftci has proposed a method to construct RBMs using I&F neuron models and to train them using an abstract model based on Neural Sampling [107]. In neural sampling framework [108], each spike is interpreted as a sample of an underlying probability distribution. Using this framework Buesing et al. provided an abstract neuron model consistent with the behavior of biological spiking neurons and can perform MCMC which is the main part of learning in Boltzmann Machine. Also they have shown the efficiency of using this abstract model in learning task in a Boltzmann Machine. In [106] Neftci et al. identified the conditions under which a dynamical system consisting of I&F neurons, can perform neural sampling and then demonstrated learning in a Restricted Boltzmann Machine using STDP synapses. This work provide an online, STDP-based learning rule to train RBMs sampled using LIF neurons. In fact they focused on the timing aspect of neural coding and used an abstract model which helped to provide a STDP implementation of learning process in RBM.

In [94] O'Connor et al. have used an abstract model of Leaky Integrate-and-Fire (LIF) spiking neuron called Siegert Neuron ([109]) to approximate the average of output firing

rates of LIF neurons. Siegert neurons have transfer functions that are mathematically equivalent to the input-rate output-rate transfer functions of LIF neurons with input spike train with Poisson distribution (Figure *3-18*).

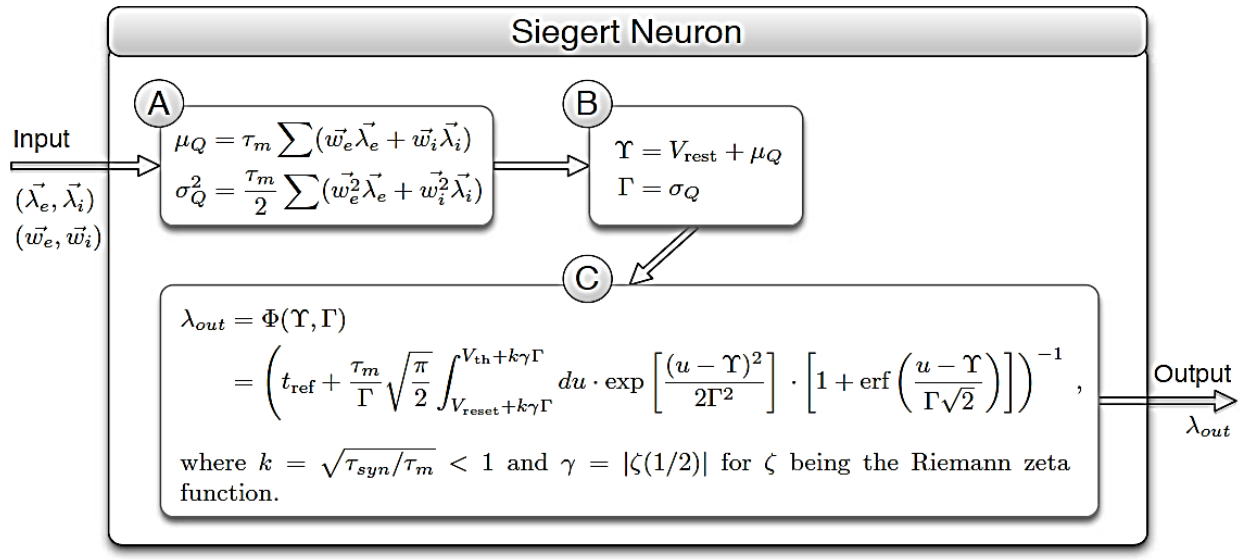


**Figure 3-18: Siegert Neuron model has a transfer function that is mathematically equivalent to input-output rate transfer function of LIF neuron with Poisson-process input [110]**

Using this abstract model for units of a Deep Belief Network and exploiting the standard *Conterastive Divergence (CD)* algorithm, O'Connor have trained a DBN offline. After training the adjusted weights are transferred to a functionally equivalent Spiking Neural Network of LIF neurons. In this article they perform learning of the network offline, rather than with spike-based learning rules.

The *Linear-Nonlinear (LN)* model is another model that can be used to describe the relation between input and neuron response as the corresponding firing rate. Indeed this model is a simplified neural response model that can map the current input to output firing rate, but it cannot describe the biological aspect of neural activities. In [111] the Linear-Nonlinear model is related to the more biological detailed models such as Leaky Integrate-and-Fire (LIF), Exponential Integrate-and-Fire (EIF) and conductance-based Wang-Buzsaki models. Using LN they provide a good estimation of the output firing rate generated to response the input signal. The NL model generally is linear filter which is cascaded by a static nonlinear transformation. [111] introduces an adaptive timescale rate model. To assess the accuracy of the model they compare the analytic driven linear filter and static nonlinearity with the numerical model. The numerical model can be determined using *Reverse Correlations* approach [77, 112]. The proposed model can provide accurate

estimation of instantaneous firing rates.

*Spike Triggered Average (STA)* is another numerical method which can be used to determine neural responses with white-noise stimuli. Indeed Spike Triggered Average and Reverse Correlations are two simpler alternative for the *White Noise Analysis* techniques which has been used in older work for obtaining neural response [112]. The STA can provides a linear estimation of the system response, then not only for a system with linear transform function the STA can provides a complete characterization, but also the STA can be used as the linear filter in LN models.

The linear nonlinear model generally can be followed by a Poisson Spike Generator block as *Linear-Nonlinear-Poisson (LNP)* model [77]. The recent block converts the generated instantaneous firing rate to corresponding instantaneous spike train (Figure 3-19).

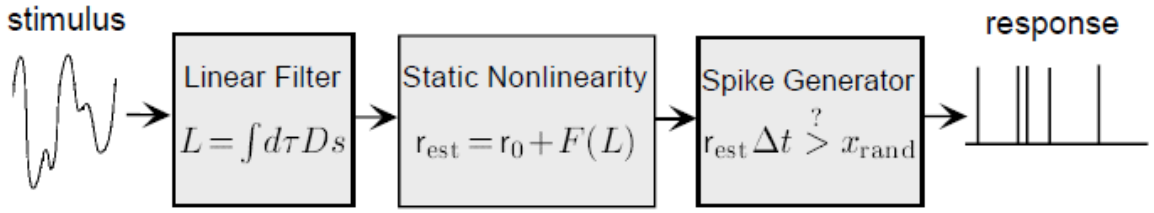


**Figure 3-19: Generating spiking respose to stimuli using LNP [77]**

Using the mentioned techniques one can simulate the answer of a network of Spiking Neurons. But in this approach the behavior of the network is simulated using analytical methods. Actually in this kind of simulation we only care about the mathematical models and using simulation to find the equation's answer. This approach does not contain any implementations of neurons or synapses. Respecting this perspective, implementation using this models is not an *online learning* and to assess the accuracy of the model in a network of spiking neurons (i.e. LIF), one needs to train the network using the behavioral models (i.e. Siegert) and then transfer the trained weight to a spiking network with same proposed conditions for the behavioral simulation (i.e. [94]).

In this study we are going to provide an online learning strategy to train RBM and DBN using LIF neurons, with focusing on *rate aspect* of *neural coding*. In the following, we will explain our methodology that we called it *Rate-Coded Contrastive Divergence (RC-CD)* for utilizing CD in a spiking RBM which help us to develop a Spike–Based Deep Belief Network. Our reasons for this selection will be discussed later.

## 3.5 The proposed approach to use CD in a spiking framework

Finding the relation between stimuli and corresponding neural response is one of the fundamental issue in neuroscience. Studying neural dynamics in *microscopic level* is too complex, because of the stochastic behavior of neuron's response to input stimuli. Therefore, as was mentioned above, the input-output transfer function for each neuron is approximated using abstract and simplified models which are not accurate as the biological models such as HH or LIF.

*Macroscopic description* is an alternative for *microscopic level* investigation. Macroscopic description for a neural network is interested in assuming a sub-population of neurons as single model and determine the properties of the whole sub-population instead of characterizing the single neurons and their interactions. Specifically *brain* as a too complex neural network is consisted of billions of neurons. The interactions between neurons are so elaborated which are not tractable using simple models of neurons. Too many efforts have been done to achieve a precise model which can explains the variety of neural phenomena [113-116]. The aim of these models is not only reducing the complexity of neural commutations, but is to propose a population model with identical characteristics which can be investigated as a single sub-population. Statistical approaches are required tools to study dynamics of such a large population. *Mean Filed* is a theory from statistical mechanics and physics which has been widely used in neuroscience researches.

Using *Mean Field Theory (MFT)* the complexity of a stochastic system can be modeled by a simpler one. A large scale model is consisted of a large number of single elements with their mutual interactions. Mean Field Theory in such a system, can approximate the total effects on a single component have been made by others. Indeed MFT can estimates the effect of the other on a single particle by an *average* to reduce the problem size. In the other words *a large scale neural activities can be investigated via MFT.*

In 1972-1973 Wilson and Cowan proposed a model [113, 114] for approximating the neural interactions using Mean Field approach. Respecting the Mean Field Theory they provide a framework for analyzing the total properties of a large network of neurons. Indeed they presented a *macroscopic* approximation of neural activates. The Wilson-Cowan equations has been used widely in computational neuroscience and many applications in neuroscience and machine learning have been instigated using mean filed theory. Very well results have been obtained using Mean Field Theory that one can find a list of some

developments based on Wilson–Cowan model in [117].

Transfer function in Wilson-Cowan model has the main role. Briefly the transfer function in their model is a function of input firing rate to output mean rate. The proposed transfer function can estimates real cortical neurons activities [117, 118]. Indeed the Wilson-Cowan model can be thought as a *firing rate model which is able to predict complex intrinsic properties of a population of neurons*. Integrating *macroscopic* models as a large scales model with more detailed biological *microscopic* models such as the Hodgkin–Huxley and Integrate-and-Fire, *Multi-scale* models have been developed [117].

Boltzmann Machine as a recurrent stochastic network model, has many interesting properties to be a good model for neural activity investigations. There are some theories about the mechanism of inference, prediction and internal representation in cortex, then respecting to the ability of Boltzmann Machine in generating internal representation, it has been studied from this point of view ([119-121]). Also Boltzmann machine as a Hopfield model, is an associative memory which is suitable to study how it works in biological related issues ([122]). Regarding the mentioned benefits, Boltzmann Machine is a proper model for neuroscience researches development.

Another influential exploitation of Mean Field Theory on neural network investigation, has been done in [52]. They provided a deterministic Mean Field approximation to replace with the longtime stochastic Boltzmann Machine learning process [123]. The proposed method in [52], which is known as *Mean Field Boltzmann Machine (MFBM)*, expressed the computationally intractable stochastic measurements of correlations with a fully factorized mean field approximations of neurons activity. Since MFBM did not has any mathematical guarantee, therefore in [124] professor Hinton shown the efficiency of [52] mathematically. Eventually, in [53] the method which is the common method today, has been expressed. Indeed the Contrastive Divergence leaning method, which is the basis of leaning algorithm in deep structures, is a Mean Field approximation which had been replaced with Gibbs sampling (which is a long time process).

Boltzmann Machine weight updating rule, considering its effect on the probability density, has positive and negative phase. Positive phase increases the probability of training data and negative phase decreases the probability of samples generated by the model. In fact in each phase we have to compute the corresponding expectations.

From another point of view, Contrastive Divergence can be interpreted as Hebbian-Anti Hebbian weight plasticity adjustment rule which updates the weights proportionally to the difference of the correlation between activities of hidden and visible units ($\langle v_i h_j \rangle$) in positive and negative phase. This approach has been called *Contrastive Hebbian Learning (CHL)* in [125]. [125] not only proved that the Mean Field approach can be applied to any continues Hopfield model (especially in Boltzmann Machine), but also explained the equivalency between Back-Propagation and Contrastive Hebbian Learning approaches when no hidden units are defined.

As was mentioned before, in negative phase (also called sleep phase) Boltzmann Machine run Gibbs sampling for a long time to reach an equilibrium point. The alternative approach to this Gibbs sampling which is expressed in [52] is a fully factorized Mean Field distribution ($Q(s) = \Pi_i m_i^{s_i}(1 - m_i)^{1-s_i}$ ). MFT provides the averaged effect of all pre-synaptic neural activities as a faster but not more accurate approximation of stochastic Gibbs sampling.

In [53] the results of using Mean Field Theory in Boltzmann Machine ([52]) have been utilized to develop a *one-step Contrastive Divergence* for *Restricted Boltzmann Machine (RBM)*.

In Restricted Boltzmann Machine (same as original Boltzmann Machine), the stochastic state of each given neuron is effected by all the pre-synaptic neurons (Figure 3-20). In [53] the approximated Mean Field value of the desired correlations ($\langle v_i h_j \rangle$) in negative and positive phase, have been expressed as product of two *sigmoidal function* of states of previous layer units and corresponding weights ((3-20)).

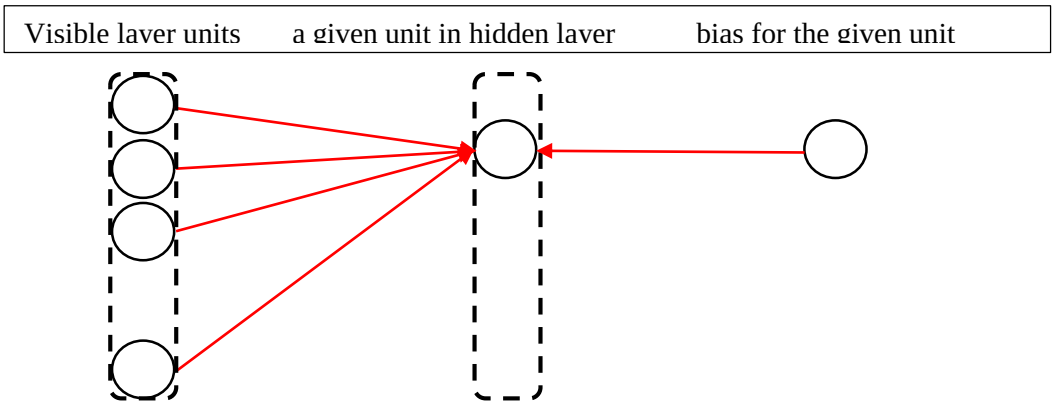


**Figure 3-20: The effect of pre-synaptic and bias units on a single neuron in RBM.**

$$\langle v_i h_j \rangle = m_i m_j$$
$$where\ \ m_i = \sigma\left(\Sigma_j w_{ij} m_j + bais_i\right)\ and\ \ \ m_j = \sigma\left(\Sigma_i w_{ij} m_i + bais_j\right) \quad (3\text{-}20)$$

Also the fantasy state of machine (running freely) is replaced by *one-step* running. Therefore practically Contrastive Divergence in RBM can be driven from difference of Mean Field approximation (instead of this expectation:$\langle v_i h_j \rangle$) averaged states of units in positive phase (data or wake phase) and negative (model or sleep) phase:

$$\Delta w_{ij} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model} = m_i^0 m_j^0 - m_i^1 m_j^1 \quad (3\text{-}21)$$

This recent equation for weight updating, is consisted of Hebbian and Anti-Hebbian terms. Indeed $m_i^0 m_j^0$ by increasing the synaptic weights is a *Long Term Potentiation* of synapses *(LTP) (Hebbian term)* and $m_i^1 m_j^1$ by increasing the synaptic weights is a *Long Term Depression (LTD) (Ant-Hebbian term).*

Respecting to the special characteristic of Boltzmann Machine (also the restricted one (RBM)), and its importance in neuroscience, we are instigated to develop a spiking version of DBN. As we know, first we need to propose a spiking model of RBM. One can find some models of spiking RBM in related work. For example in [126] the authors developed a temporal version of BM, but indeed in this work no model of biological detailed spiking neurons can be found. In [106] using *Neural Sampling* method a STDP based version of Contrastive Divergence has been proposed. Indeed [106] respecting the stochastic properties of neural networks, develop a framework to train a RBM of Leaky Integrate-and-Fire neurons using STDP. Another work that implement a DBN using LIF neurons is [94] that uses the *Siegert* model.

At this point it seems important to explain some significant keys of our goal. As was mentioned before an abstract model (i.e. the *Siegert model [109]*) is a set of mathematical equations which can explain the firing out of a given neuron dealing with other pre-synaptic neurons in a population. These models can be driven by inversing the interval time respecting the statistical methods to approximate the transfer function of a given model. In this dissertation implementing an abstract model of biological circuits (i.e. LIF) is not our care. Indeed we are interested in simulating a model that can be implemented as a population of detailed biological models (i.e. LIF or HH). There is a big difference between this model and an abstract model. An abstract model requires a specific hardware for computing the

related equations and we cannot find any aspect of biological interactions between units.

Considering the biological models, information should be coded using spikes not real values and neurons required to be interacted through these spikes. The recent property can provide a low power implementation respecting using discrete small electrical pulses and also non-Von Neumann architecture which leads to less instruction and data transferring from and to memory [14, 15]. Such an architecture which can compute in parallel is a *Neuromorphic Accelerator*. (Indeed we are interested in implementing a Neuromorphic Accelerator in future work using Memristor cross-bars.)

After clearing our aim of this study, we have to prepare the necessary tools we need to develop our spike-based model of Deep Belief Network. First, we need a methodology to encode and decoding the value and the spike trains. Because of using MNIST benchmark in our work, it's important to introduce it. Then we can develop an *online learning rule* to train the spike-based model of Restricted Boltzmann Machine that we called this rule *Rate Coded-Contrastive Divergence (RC-CD).* And finally we can present the spike-based Deep Belief Network.

### 3.5.1 Generating Spike-Train from traditional machine learning benchmark

Benchmarks and datasets have important role in evaluation of researchers proposed method for machine learning tasks. There are some standard datasets for evaluating machine learning algorithms in different types of applications. One of the more popular dataset in image processing tasks, is MNIST which is used in our work. Here at first, we introduce this dataset.

#### 3.5.1.1 The MNIST database of handwritten digits

There are some famous databases for examining the accuracy and performance of algorithms proposed by machine learning researchers and professionals. These databases are in various fields. For example if someone needs to examine his new approach in neural networks for recognition of facial expressions, he needs a lot of train data and should has some data for test. Also if he wants to compare his work with others, he must use a database similar to others. For this problem there are some standard databases. One of this databases in handwritten digits recognition is MNIST. MNIST is a standard and large database of handwritten digits.

MNIST dataset has been widely used as a benchmark for testing classification

algorithms in handwritten digit recognition systems. MNIST is an abbreviation for Mixed National Institute of Standards and Technology database. This database is created by "re-mixing" the original samples of NIST's database. The database has two parts: training samples that was taken from American Census Bureau employees and the test samples that was taken from American high school students.

The original NIST's database is too hard, therefore the MNIST database was constructed from NIST's Special Database 3 and Special Database 1 which contain binary images of handwritten digits. The samples that was taken from American Census Bureau employees (training database), was very cleaner than the samples that was taken from American high school students (test database). It can makes dataset dependency for our learning algorithm (because of more readability of training set than test set). Therefore ''re-mixing'' NIST's datasets was necessary.

By combining of 30,000 samples from first dataset and 30,000 samples from second dataset, the Mixed NIST training set was created. Also 60'000 test samples was collected to constitute the test dataset, but only 10,000 of patterns is now available on MNIST webpage. The first 5000 examples of the test set are taken from the original NIST training set. The last 5000 are taken from the original NIST test set (The first 5000 are cleaner and easier than the last 5000).

The MNIST is widely used for training and testing in the field of machine learning. In some papers the training set was extended by adding some distortions (random combinations of scaling, shifts, adding some noise etc.). In [2] there are the results of some methods that have been tested with MNIST dataset.

The MNIST database is available on MNIST webpage [127] and contains of 60,000 examples for training set and 10,000 examples for test set. This database includes 4 files:

train-images-idx3-ubyte.gz: training set images (9912422 bytes)

train-labels-idx1-ubyte.gz: training set labels (28881 bytes)

t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)

t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

In MNIST database the original black and white images from NIST were size normalized to fit in a 20x20 pixel box. The resulting images contain grey levels as a result of the anti-aliasing technique used by the normalization algorithm and the images were centered in a 28x28.

The data of all images in each database are joined in a single file and are stored in a file format that called IDX. IDX is a file format for sorting vectors and multidimensional matrices of various numerical types. The basic format is:

magic number
size in dimension 0
size in dimension 1
size in dimension 2
.....
size in dimension N
data

The magic number is an integer that stored in the MSB first format, then for users of Intel processors and other low-endian machines must flip the bytes of the header.

The magic number format and other technical issues is described in our report [128]. Finally we extract the images from MNIST dataset and reorganized them in vectors of image. In Figure 3-21 you can see some exported images from MNIST which is plotted in MATLAB.

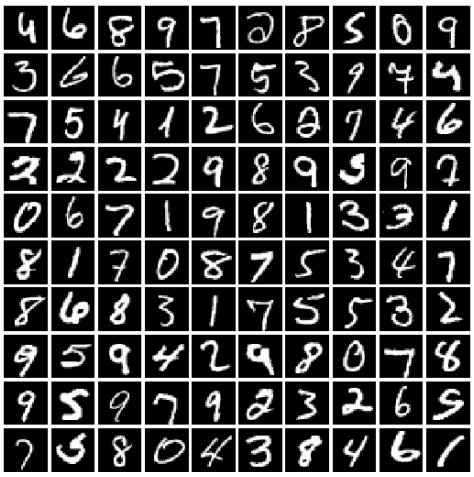

**Figure 3-21*: 100 digits from MNIST***

### 3.5.1.2 The dataset problem statement and proposed methods

As was disused in previous sections, there is gap between machine learning methods and spiking models of neural networks. Consequently, this challenge is remained in usage

of traditional frame-based images in existing datasets. Since Spiking Neural Networks use spikes for communication between their neurons, then the information transferring between neurons is just possible through spikes. In this communication, since spikes of a given neuron are same, the form of the spike does not carry any information and the number and the timing of spikes are important. Therefore if we want to use these benchmarks to evaluate our work, we need to extract proportional spike sequence from them, considering, the proper number of spikes and interval between spikes.

There are some ways to overcome this problem, but on the other hand, there are limited public method that can be used by any SNN researcher. Indeed if any one cannot used these public methods, he must to provide his personal method. For example in [101] authors, presented each images to the network for 350 ms in the form of Poisson-distributed spike trains, to encode the input to spike trains, and provide input firing rates between 0 and 63.75Hz and repeat the process until at least five spikes have been fired during the entire time the particular example was presented. Authors of [100, 129], present the input as asynchronous voltage spikes using several coding schemes. In [94, 96] to convert the static images into events, the images are vectorized into neuron addresses and spikes were generated with probability proportional to the intensity of the pixel.

According to our investigations, in many perfect applications of SNN for image processing tasks, these stimuli generated directly from a spiking "retina" that naturally present data as asynchronous. In [130], in response to the visual input, spikes produced by the DVS .Also, in [131], A random subset of the spikes emitted by the DVS are mapped to 128 hidden layer neurons.

"Retina" [132], as the common way in Neuromorphic community, is an AER bio-inspired image sensor which measures visual information not based on frames from scenes and generates corresponding spike sequences. Address-Event Representation (AER) is a Neuromorphic communication protocol for spiking neurons. In AER, spikes are represented as digital addresses and leading to the address-event representation of images. In [132] a Asynchronous Temporal Contrast Vision Sensor was proposed which can independently generate spike events from each pixel relative intensity changes. Dynamic Vision Sensor (DVS)( Figure 3-22), the ready for used Asynchronous Temporal Contrast Vision Sensor, used technology that works like retina. Instead of sending entire images, only the local pixel-level changes and the time of changes are transmitted (Figure 3-23).The output of the sensor

a stream of events at microsecond time resolution.

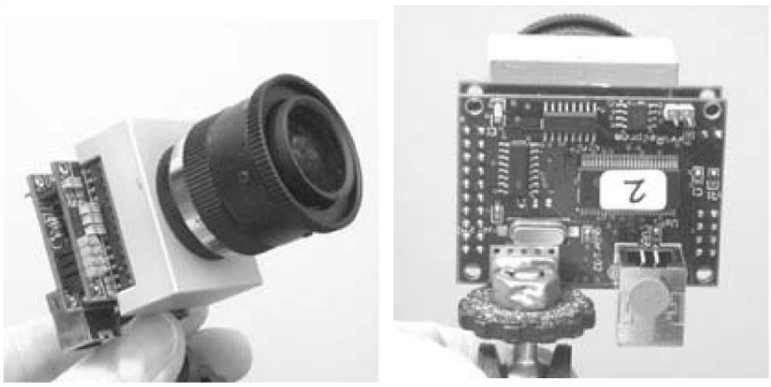

**Figure 3-22: Dynamic Vision Sensor (DVS) - asynchronous temporal contrast silicon retina**

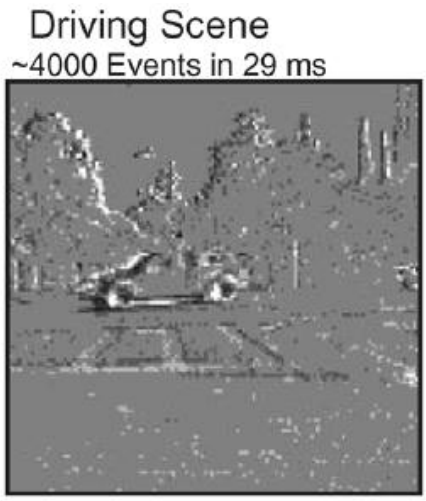


**Figure 3-23*: Example images data from the vision sensor***

These Neuromorphic vision sensors is not publicly available, so we need to use some other tricks. As was expressed in [26], a spike train is a sequence of spikes generated by a single neuron. A spike train can be defined as a sequence of spike's times. Any neuron in a neural network, receives input spikes from their presynaptic neurons (Figure 3-24). This chain of events can occur at regular or irregular intervals. Some evidence for neuronal variability and spike-train irregularity were reviewed in [26] (Recording spike trains from single neurons using electrophysiological methods). In the cortex, action potentials timing are irregular [133] and it is not periodic. Each irregular spike sequence can be described as stochastic sequence generated by a process. Here, we assume that the generation of each spike is independent of all the other spikes. Respecting to this independency, the spike train would be completely described through a Poisson process. It is notable that, some aspects of neuronal dynamics, can break the independent spike assumption [28]. Also we postulate that

the firing rate for each synapse is constant over time. This means we assumed the Poisson process is homogeneous. According to [134], Poisson processes can describe the irregularity of spike times. The simplest stochastic description of neuronal firing is a Poisson process. Indeed Poisson process is the simplest stochastic description of neuronal firing. As it was shown in [26], since each spikes in a Poisson process are independent, then Poisson firing cannot account for refractoriness. The using of renewal processes, which is depends on the time of the last spike is suggested. (Here for simplicity, we use Poisson process.)

Here we used Poisson distribution for generating spike trains of images of MNIST dataset to extract a spike-based dataset from a frame-based dataset respecting to intensity of each pixel of images.

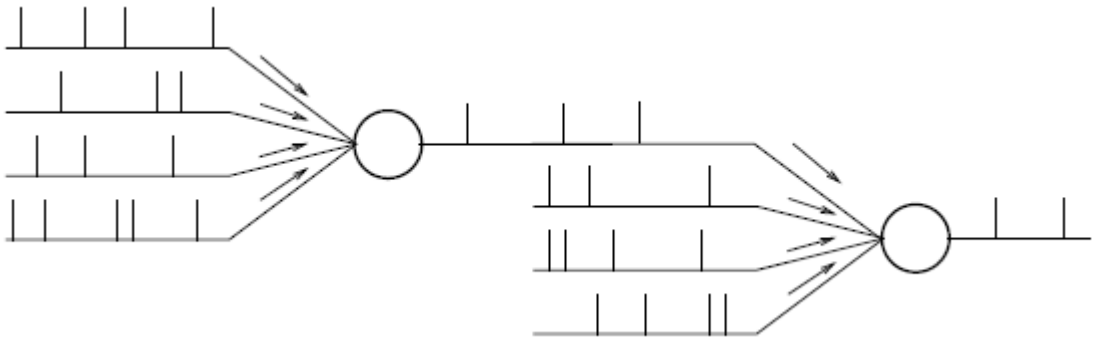

**Figure 3-24: Each neuron receives input spikes from presynaptic neurons [26]**

There are two commonly used procedures for generating homogeneous Poisson spike trains [133]. The first one is driven from the relevance of Poisson and Exponential distributions. If we choose Inter Spike Intervals (ISI) randomly from a given exponential distribution, cumulating these ISIs can generate continues sequence of spike times.

According to our assumption -the generation of each spike is independent of all the other spikes-, we also assume the spike generation is only depends on instantaneous firing rate. In our simulation we postulate any pixel's density of MNIST digits, as the instantaneous firing rate of stimuli which is clamped on inputs neurons during the simulation time. Indeed we refer to intensity of pixels as the probability that a spike occurs within an interval. Then we can use the MNIST digits vector as a probability vector. If we dived the simulation time into some time periods $\Delta T$, which in each period maximum number of spike is 1, then we have n bins (3-22):

$$Bins_{Number} = \frac{Simulation_{Time}}{\Delta T} \qquad (3\text{-}22)$$

The firing rate $r$ (pixel's density) determines the probability of firing a single spike in any $\Delta T$. The probability of occurring a single spike in any $\Delta T$ is (3-23):

$$r\Delta T \qquad (3\text{-}23)$$

In homogeneous Poisson process, rate of spike generation in any $\Delta T$, is only depended on stimuli intensity in $\Delta T$, respecting to constant intensity of pixels during the simulation, for all intervals of simulation time ($\Delta T$), we have constant spike generation probability as the mean firing rate.

Therefore according to this fact that the probability of firing a spike during a short period of time (i.e.: 1ms) is $r\Delta T$, using a simple sampling procedure (Algorithm 3-1), each generated spikes will be assigned to discrete time.

*Algorithm 3-1*: Poisson spike train generation

1: ***Find*** *M as the number of bins:*
$M = \frac{Simulation_{Time}}{\Delta T}$

2: ***Generate*** *M uniform random numbers between 0:1 as sequence:*
$X = \{x_{rand}\}$

3: ***For*** *each bins:*

***if*** *($r\Delta T > X_i$)*

***generate*** *1 spike*

***else***

*do nothings*

It was proven in [77], when the probability of having 1 spike in $\Delta T$ is $r\Delta T$, then the probability of having n spikes in total time (T), $P_T[n]$, is a Poisson distribution.

To evaluate the accuracy of the proposed method for spike generation in Poisson distribution, respecting to definition of Poisson process, we have to setup a simulation. In this simulation we generate some spike trains from a given pixel density, which is equal to sampling a specific neuron in a certain time interval for many trials. Using Algorithm 3-1, we can generate some spikes and count the number of spike in each trial. Finally we have to evaluate the given sequence of firing numbers in different trials (3-24).

$$Numeber\ of\ spikes\ in\ k\ trials = \{n_1, n_2, \dots, n_k\} \qquad (3\text{-}24)$$

One of the best measurement for evaluating a given distribution to see if it is a Poisson

process is *Fano Factor* (3-25). Since for a Poisson distribution variance and mean of the event count are equal, then the *Fano Factor* for a Poisson process is one.

$$Fano\ Factor = \frac{Variance_{Dist}}{Mean_{Dist}} \qquad (3\text{-}25)$$

Figure 3-25 shows a chosen digit from MNIST for spike train extraction using the Algorithm 3-1. The instantaneous firing rate is chosen from pixel of the vector containing the pixels value of Figure 3-25.

**Figure 3-25: One MNIST digit for extacriting corespoding spike trains for each pixels**

We repeat the spike generation for 1000 times and generate 1000 spikes trains. Counting the number of spikes in each trail, we have a vector of 1000 number, which we have to estimate its distribution and *Fano Factor*. We use Matlab program for our simulation. In Figure 3-26 you can see the histogram of the given vector and the fitted distribution.

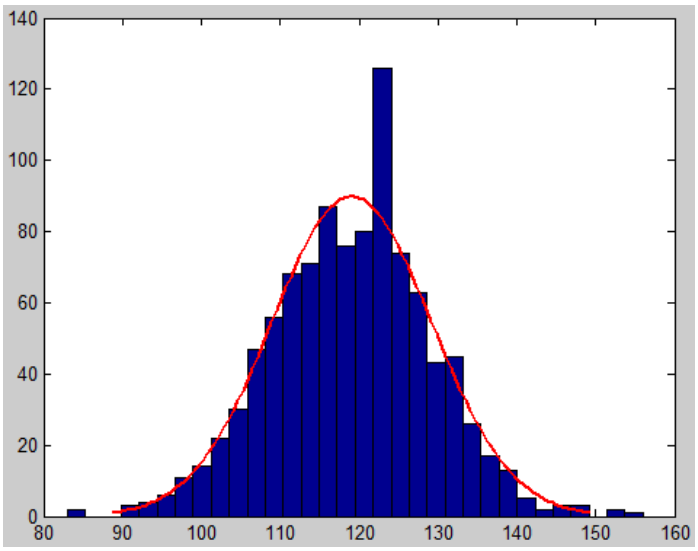


**Figure 3-26: Histogram of spike train simulation**

The estimated Poisson $\lambda_{hat}$ parameter (is equal to r) using Matlab is 119.569, which is very close to the chosen pixel density. You can find other parameter in Table 3-1. $\lambda_{hat}$ is the maximum likelihood estimate (MLE) of the parameter of the Poisson distribution, λ, for the given data.

Table 3-1: Estimated parameters for the spike train simulation

| Estimation Parameter | Estimated value |
|---|---|
| $\lambda_{hat}$ | 119.569 |
| Variance | 112.9142 |
| Mean value | 119.5690 |
| Fano Factor | 0.9443 |

As can be seen in Table 3-1, the estimated *Fano Factor* is close to 1. We also use *Easyfit* software to fit probability distributions to given vector of number of spikes in 1sec (Figure 3-27).

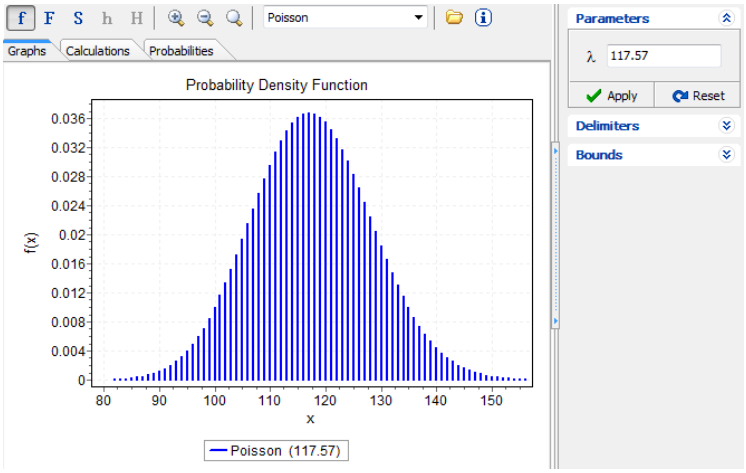


**Figure 3-27: Easyfit distributin estmation**

Using this method, we generate an event-driven dataset of MNIST called evt-MNIST. We convert all the test and train images of MNIST dataset and upload and share it on https://github.com/MazdakFatahi/evt-MNIST. These are two videos which playback the spike generation corresponding to pixels rate. You can see pixels with more density (the righter pixels) have more spikes/sec. We assumed 100ms for pixels representation. Indeed we used the pixels density as the intensity of stimuli which is clamped on the inputs of our SNN during of simulation. The maximum rate for completely white pixels is 1000 Hz (100 spikes in 100ms). Indeed we prepared a AER-like data set which is consisted of 60000 matrix for training images and 10000 for test in size of 784*100 for each matrix. We assume each pixels of any pixels as an input stream for triggering the corresponding neuron in our proposed architecture in next chapter. Each input stream is a spike train which is generated using described algorithm. Therefor we have a matrix of 784*100 logical variables. For each pixel we have a 1*100 spike train of true and false (true = spiked /false=no things), which

as was shown the number of spike in each trails have Poisson distribution. It's clear that for a pixel which is brighter, more "true" in spike train is generated for. In Figure 3-28, we can see the generated spikes for a sample image. The spike streams will be integrated in receptive neurons (Figure 3-24).

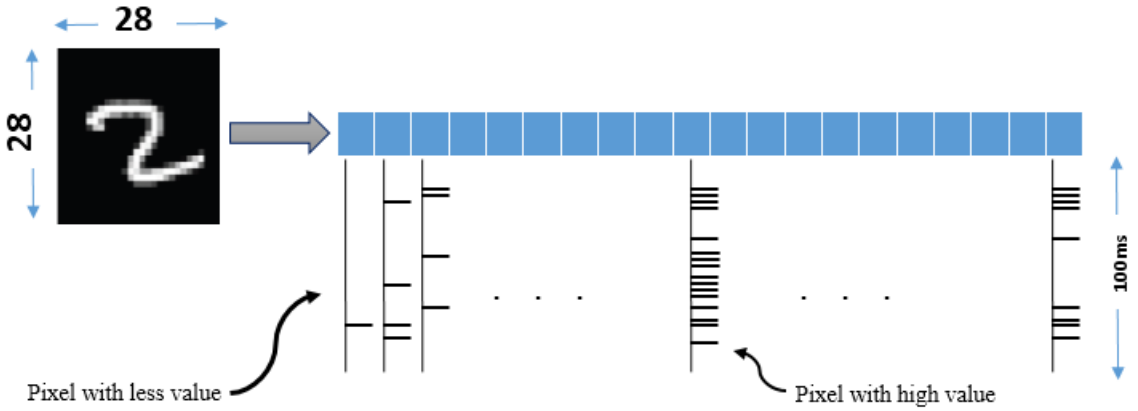


**Figure 3-28: Converting static images to dynamic spike trains: It's clear the pixels with less value have less corporation in learning process then in spike t streams the have les spikes and conversely about pixels with high value.**

When we use these generated spikes train as input for a Spiking Neural Net, the neurons which receives this streams of more spikes, will be more triggered. In Figure 3-29 we depicted the effect of number of spikes which is directly depended on pixels density and the presentation time of each images. We can see the pixels with lower intensity, in contrast to pixels with higher value, in same time, are less bright.

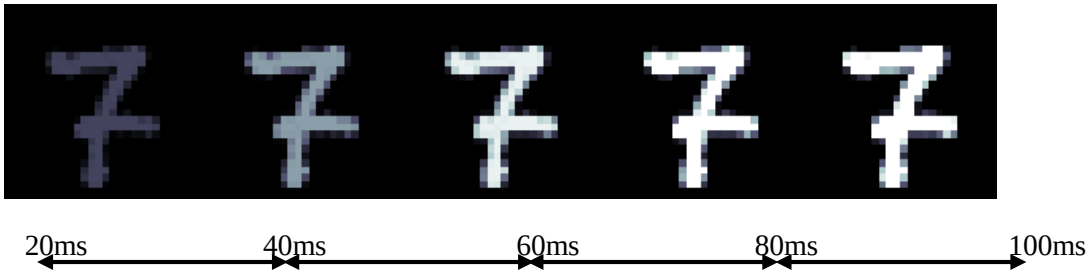


**Figure 3-29: Effect of number of spikes: When an input triggered continuously, the receptor neurons can be more determinative in learning process**

Similarly in Figure 3-30 we can see the effect of randomness in spike generation. Indeed pixels with higher value have more spikes and can be seen mostly "on". The pixels with middle value, proportional to their pixel density are "on" and "off" time to time, but pixel with very low density are mostly "off". This issue is illustrated in the raster-plot of spike trains of 784 neuron (corresponding to number of pixels in a MNIST image) in 1000ms (Figure 3-31). The mostly "on" pixels are located in the center of images, consequently we can see denser spike trains in the middle of Figure 3-31.

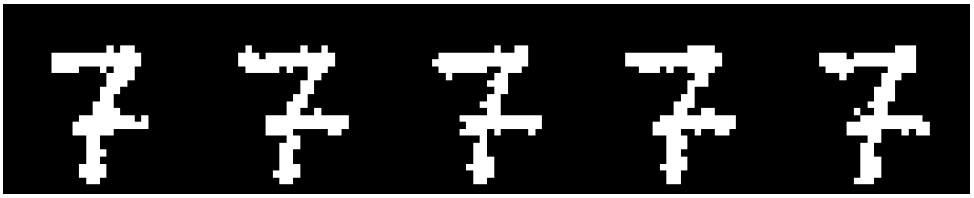

**Figure 3-30: The effect of pixel's density in spike generation**

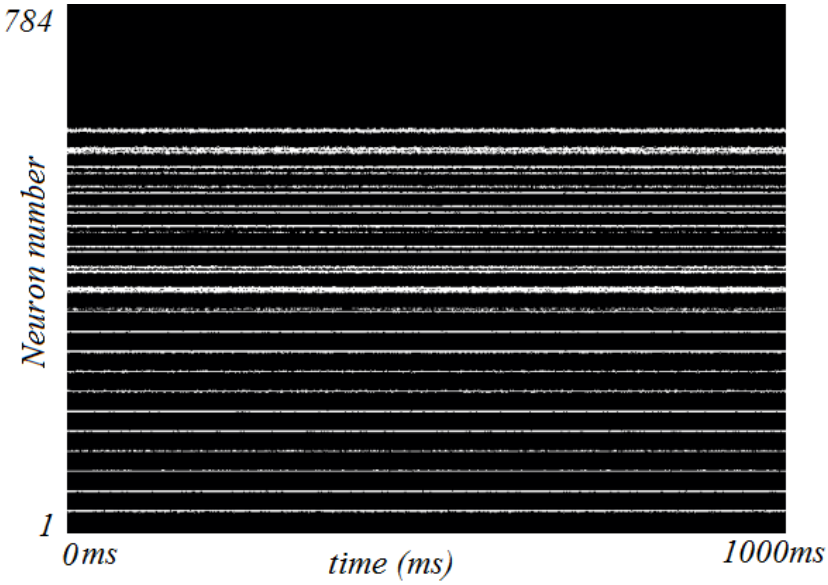


**Figure 3-31: Raster-plot of spike trains for a given image**

### 3.5.2 Spike-Based Restricted Boltzmann Machine

In 3.2 we talked about rate and time coding aspects of neural coding. We know the interpretation of neural coding is depended on our observations [89]. Also we discussed about the Mean Field Theory as a statistical tool which can explain the neural activity of a population of identical spiking neurons by finding the gain function (transfer function) of input firing rate to output mean firing rate for each sub-populations and assigning the function to each individual neuron.

In this section, regarding the mentioned issues, using rate aspect of neural coding we develop a model of RBM which is consisted of Leaky Integrate-and-Fire neurons. Our model uses the Hebbian Anti-Hebbian property of Contrastive Hebbian Learning which is used in Boltzmann Machine. But there are some difference here:

- Despite of original Boltzmann Machine and Restricted Boltzmann Machine which are usually consisted of stochastic binary units, our model is consisted of LIF neurons which interact with each other using spike trains.

- Respecting to long time Gibbs sampling process in CHL, we prefer to use the fast method which is based on presenting one-step Mean Field approximation of desired expectations in Contrastive Divergence. But we need a method to approximate the expectations in Contrastive Divergence updating rule.

#### 3.5.2.1 Rate Coded Contrastive Divergence: An online learning rule for spike-based RBM

As we know in [53] the results of using Mean Field Theory in Boltzmann Machine ([52]) lead to *one-step Contrastive Divergence* for *Restricted Boltzmann Machine (RBM)* which can approximate the expectations terms ($\langle v_i h_j \rangle$) in negative and positive phase using *sigmoidal functions* ((3-20)).

This approach is based on the macroscopic observation of the total network where the network is consisted of identical neurons. Identical means that all the neurons have same parameters and same type in a population. Regarding the correctness and computational of this method, we are interested in a situation when the neurons are not same, at least in their parameters. Consequently we can say, we are looking for a *Multi-Scale* model for spiking RBM.

Variety in parameters needs to microscopic observation of neurons in a given population. In this study we use the result of using Mean Field approximation in Contrastive Divergence which reduced the original BM learning rule to a difference between two factorized mean field values. But respecting to violating the hypothesis of using identical neurons, we cannot use the approximated Mean Field values.

The intrinsic stochastic behavior of neural activities and the variety of observed results for identical experiences in neural networks trial-to-trial, lead researchers to interpreting neural activities as samples from an underlying probability distribution [135]. In the other words, each configuration of states of neurons, in each layer, can be assumed as a value of a stochastic variable. Figure 3-32 shows a given layer of neurons which has some triggered neurons. Respecting to the states of neurons, *X* can be defined as a random variable (3-26) which in each iteration can expresses the configuration of the layer.

$$X^{(i)} = \left(x_1^{(i)}, x_2^{(i)}, x_3^{(i)}, \dots, x_l^{(i)}\right) \qquad (3\text{-}26)$$

$$and\ x_j^{(i)} = \begin{cases} 1 & state = Fired \\ 0 & otherwise \end{cases}$$

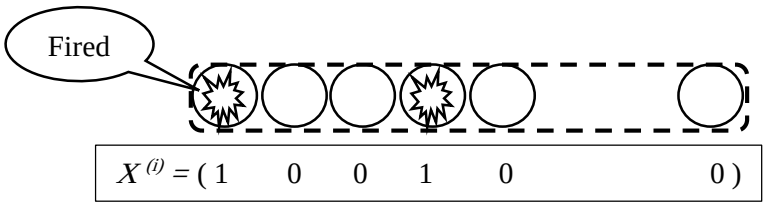


**Figure 3-32: Neural activities can be interpreted as a stochastic variable**

To compute the LTP and LTD values in positive and negative phases of contrastive divergence, we need some preliminary things:

- Respecting to Contrastive Divergence updating rule we need a method to compute the *expected values* in positive and negative phase to obtain the LTP and LTD.
- Considering the expectation definition to compute the expected values (3-27), we need to know the *firing probability* of each given neurons.
- State of each given neuron is a stochastic variable which is depended on its pre-synaptic states. Therefore we need a method to *update* the state of neurons in each layer.
- Considering using MNIST dataset to evaluate the accuracy of our model, we need a method to convert the frame-based images to *streams of spike trains*.

$$\mathbb{E}[f] = \sum_{n=1}^{\infty} f(x_i)p(x_i) \qquad (3\text{-}27)$$

The required steps are related to each other. The first step for updating all values is converting images to spike trains according to that is described in 3.5.1. As arriving a spike at each given neuron, the neuron membrane voltage will changed and may causes to generating new spike. Each emitted spike will propagate through the network. The spike's effect on other post-synaptic neurons will update the state of post-synaptic neurons respecting to incoming spikes. Therefore the first step in updating process is depended on the characteristic of LIF neurons. The second one is to determine the conditional firing probability of each neurons respecting the previous step of updating the state of neurons, and finally we need a method to utilize the computed probabilities for evaluating the required expected values to adjust weights. Accordingly in the following we explain each step of the explained scenario.

#### 3.5.2.1.1 Neuron characteristics and state updating

The Leaky Integrate-and-Fire model as the simplified neuron model cannot explain

many features of real biological neurons, but this model can precisely imitate the spike generation of neurons in proper time of events. Indeed, integrate-and fire model is a perfect model of spike generation mechanism and also respecting to the simplicity of the model needs less computational operations ([26, 79]).

In each observation the states of neurons in a given layer is depended on the pre-synaptic activities. Then practically, we need to firstly determine the states of pre-synaptic neurons and according to the obtained vector of states $\boldsymbol{y}$, the membrane voltage of each post-synaptic neurons can be computed respecting to Leaky Integrate-and-Fire equation (3-28).

$$\tau_m \frac{du}{dt} = RI(t) - u(t) \qquad (3\text{-}28)$$

$R: membrane\ resistance$

$I(t): total\ input\ current$

$u(t): membrane\ voltage$

Dividing the total time of an observation *T*, to *k* equal $\Delta t$ ($T = k\Delta t$), such that each neuron in $\Delta t$ is fired or not (maximum number of spike in each $\Delta t$ is one), then $s_j(\Delta t)$ can describes the state of j[th] neuron in each layer during $\Delta t$ (3-29).

$$s_j(\Delta t) = \begin{cases} 1 & u_j > v_{th} \\ 0 & otherwise \end{cases} \qquad (3\text{-}29)$$

$v_{th}: threshold\ voltage$

$u_j: membrane\ voltage\ of\ j^{th} nouron$

Consequently total pre-synaptic input current to a given neuron in next layer in $\Delta t$, respecting to the link effects (weights of each connection from pre-synaptic neurons to the given post-synaptic neuron), can be shown as:

$$I_i = \Sigma_j (s_j(\Delta t) w_{ji}) \qquad (3\text{-}30)$$

$I_i: total\ input\ current\ to\ i^{th}\ post-synaptic\ neuron$

$s_j(\Delta t): state\ of\ j^{th}\ pre-synaptic\ neuron$

Assuming membrane resistance R=1, the membrane potential of i[th] post-synaptic neuron is:

$$u_i = RI_i = 1 * \Sigma_j (s_j(\Delta t) w_{ji}) \qquad (3\text{-}31)$$

$I_i: total\ input\ current\ to\ i^{th}\ post-synaptic\ neuron$

$s_j(\Delta t): state\ of\ j^{th}\ pre-synaptic\ neuron$

Respecting to the recent equation, in each $\Delta t$ of $T$, updating neurons in a given layer, is depended on determining the state of neurons in previous layer. Therefore in each observation the effect of the spike propagation should be tracked in each $\Delta t$ from first layer to last one.

#### 3.5.2.1.2 Model Architecture

Respecting to the phase of weight updating in Contrastive Divergence rule, we implement two connected structure such that the first one computes the positive phase and the second one is for computing the negative phase. We called the proposed structures *positive machine* and *negative machine* (Figure 3-33). At the end of each trail respecting the computed expectations, $\Delta w$ can be computed (3-21). The visible layer respecting to the MNSIT image size has 784 corresponding neurons and the second layer has 500 neurons.

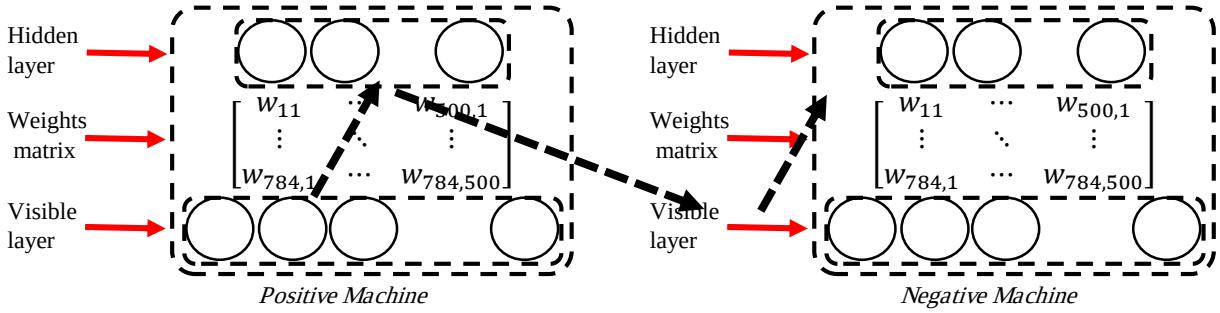


**Figure 3-33*: Positive and Negative Machines and spike tracking in network***

Considering the dynamic characteristics of the neurons, the state of all neurons in all layers, are changing continuously in time. The membrane voltage when there is no input spike will be decreased according to time constant $\tau_m$ and when there is some inputs not only the membrane voltage will increased, but if it can reaches higher than threshold voltage, the given neuron can send a spike. Therefore it's important to observe the entire of the network at a moment to investigate the internal interactions. Dividing the time by $k$ allows us to investigate the state of network entirely in each $\Delta t$. Indeed we assume that $\Delta t$ is the smallest slot of time in which the network can be updated. During $\Delta t$ we assume the input stream is stopped to all the state of neurons can be determined. In the other words, at the end of each $\Delta t$ we capture a snapshot of the network entirely. At the next $\Delta t$ the input layer of *positive machine* will be fed by the second column of spikes of the spike stream

(Figure 3-34). Indeed total training time for a given input image is deived to *k* independent trails as long as $\Delta t$.

It is important to be noticed that in each $\Delta t$, as a given time window, only one column of spike stream will be used in network. This first column will be send to hidden uints in positive machine. After updating this layer the emitted spikes will be propagated into the visible layer of negative machine and finally the results will be tracked to hidden layer in negative machine (the dashed black arrows in Figure 3-33). This assumption leads us to determine *state of all neurons in all layers*. Also it's important to be cautious about the length of $\Delta t$. Indeed we have to consider all the delayes during updating all layers which is consisted of *synaptic delays* and processing delay or *membrane delay*. Therefore in each $\Delta t$ each neuron only can spikes for one time; because firstly: in each $\Delta t$ neurons of each layer will be checked just for one time if membrane voltage reached to the threshold, secondly: each spiked neuron will be reset to $v_{reset}$ after firing and thirdly: the fired neurons after each spike will be silent for a while $\tau_{refractory}$. This time is the *refractory time* for each biological neurons as the membrane voltage of neurons despite of input currents will not increased. Also we have to be cautious until the end of $\Delta t$, new input will not be used.

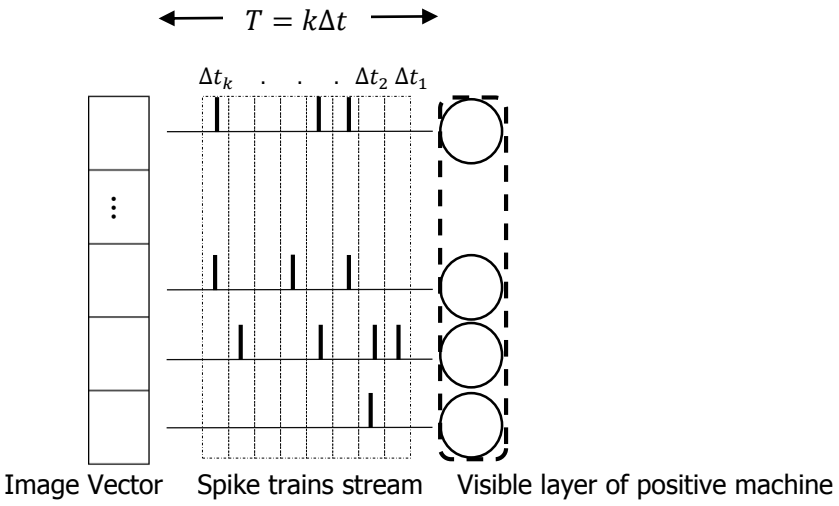


**Figure 3-34: Input spike train will be fed to the network in separated Δt**

#### 3.5.2.1.3 Computing spike probabilities numerically

In each incoming $\Delta t$ the membrane voltage of neurons respecting the total elapsed time since the last state ($m\Delta t$), and the previous snapshot of network can be determined. As is illustrated in Figure 3-20, and using the equations (3-29), (3-30) and (3-31) for updating

the LIF neurons and considering the described procedure for spike tracking, after $k\Delta t$ for each neuron the number of spikes during *T*, can be computed (obviously by recording the firing history of each neuron during each $\Delta t$). Respecting to the dependency of the state of each given post-synaptic neurons to the state of all pre-synaptic ones (Figure 3-20), we can express the firing probability as a conditional probability (3-32).

$$p(s_{post} = 1 | \boldsymbol{s_{pre}}) = \frac{n_{s_j}}{k} \quad (3\text{-}32)$$

*where*:

$s_{post}$ : *state of the given post − synaptic neuron*

$\boldsymbol{s_{pre}}$: *states vector of all pre − synaptic neurons*

$n_s$: *number of emited spikes of nueron j during T*

$k$: *total number of time windows* $(\Delta t)$

Therefore for each layer we can write same equations:

$$p(v_{pos} = 1) = \frac{n_{s_v}^{+}}{k} \quad (3\text{-}33)$$

$$p(h_{pos} = 1 | \boldsymbol{v_{pos}}) = \frac{n_{s_h}^{+}}{k}$$

$$p(v_{neg} = 1 | \boldsymbol{h_{pos}}) = \frac{n_{s_v}^{-}}{k}$$

$$p(h_{neg} = 1 | \boldsymbol{v_{neg}}) = \frac{n_{s_h}^{-}}{k}$$

*where*:

$v_{pos}$ : *state of a given neuron in visible layer of positive machine*

$h_{pos}$ : *state of a given neuron in hidden layer of positive machine*

$v_{neg}$ : *state of a given neuron in visible layer of negative machine*

$h_{neg}$ : *state of a given neuron in hidden layer of negative machine*

$\boldsymbol{v_{pos}}$: *states vector of all neurons in visible layer of positive machine*

$\boldsymbol{h_{pos}}$: *states vector of all neurons in hidden layer of positive machine*

$\boldsymbol{v_{neg}}$: *states vector of all neurons in visible layer of negative machine*

$n_{s_v}^{+}$: *number of emited spikes of a given nueron in visible layer of positive machine du*

$n_{s_h}^{+}$: *number of emited spikes of a given nueron in hidden layer of positive machine du*

$n_{s_v}^{-}$: *number of emited spikes of a given nueron in visible layer of negative machine du*

$n_{s_h}^{-}$: *number of emited spikes of a given nueron in hidden layer of negative machine du*

$k$: *number of time windows* ($\Delta t$)

It is obvious that the number of input spikes in *k* trials, is directly related to the under laying Poisson distribution of generating spike trains from MNIST images.

#### 3.5.2.1.4 Computing the expected values

Considering the Contrastive Divergence equation (3-34), it's required to compute the expectations of positive and negative phase.

$$\Delta \boldsymbol{w} = \langle \boldsymbol{vh} \rangle_{data} - \langle \boldsymbol{vh} \rangle_{model} \tag{3-34}$$

Without using the Mean Field approximation we have to compute the desired expectations of state of neurons numerically.

When expectations are too complex to be evaluated exactly using analytical methods, the numerical sampling methods, which are based on *Monte Carlo* techniques, can be used ([3]). As we know Gibbs sampling which is the original approach in CD algorithm, is a common method to estimate the expectation of $\boldsymbol{vh}$ in RBM training process. Respecting to equation (3-35), when $f(x)$ is a given function and $p(x)$ is an under laying distribution, evaluating the expectation of $f(x)$ requires a long time process:

$$\mathbb{E}[f] = \int f(x_i) p(x_i) \tag{3-35}$$

As was mentioned in chapter 11 of [3] the sampling methods are based on obtaining a set of samples drawn independently from an under laying distribution. This allows the expectation to be approximated by a finite weighted summation. Indeed when we have a finite drawn samples from a given distribution, sampling leads us to deal with the expectations as an average over drawn samples [3] (3-36).

$$\mathbb{E}[f] \simeq \frac{1}{N} \sum_{i=1}^{N} f(x_i) \tag{3-36}$$

In sampling estimation (3-36) *N* is the number of samples drawn from the under laying

distribution.

Considering (3-26) for interpreting state of neurons in each layer as a random variable, and also according to the assumptions for computing firing probability of neurons, now we assume $\mathrm{P}^{(n)} = (\rho_1^{(n)}, \rho_2^{(n)}, \rho_3^{(n)}, \dots, \rho_l^{(n)})$ as a random variable to describe the firing probability of each layer as random stochastic vector. In this equation $l$ is the number of neurons in a given layer of network, $\rho_l^{(n)}$ is the firing probability of $l^{th}$ neuron in the given layer in n$^{th}$ observation. In each phase we need to evaluate the expected value of $\boldsymbol{vh}$ where this is the product of two vectors. As we know this product is the result of derivation operation in equation (2-20). $\boldsymbol{v}$ and $\boldsymbol{h}$ are the probability vectors (random variables) of visible and hidden layers. $\boldsymbol{vh}$ as product of two vectors can produce all $v_i h_j$ as needed elements in visible-hidden configurations (3-37).

$$\boldsymbol{vh} = \begin{bmatrix} v_1 h_1 & \cdots & v_1 h_{500} \\ \vdots & \ddots & \vdots \\ v_{784} h_1 & \cdots & v_{784} h_{500} \end{bmatrix} \tag{3-37}$$

According to (3-36) and (3-37) we have:

$$\mathbb{E}[\boldsymbol{vh}] \simeq \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{vh})^{(n)} = \frac{1}{N} \left( (\boldsymbol{vh})^{(1)} + (\boldsymbol{vh})^{(2)} + \cdots + (\boldsymbol{vh})^{(N)} \right) \tag{3-38}$$

Where $(\boldsymbol{vh})^{(n)}$ is the obtained matrix of $v_i h_j$ in n$^{th}$ observation. Finally we can illustrate the desired expectation as:

$$\mathbb{E}[\boldsymbol{vh}] = <\boldsymbol{vh}> \simeq \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{vh})^{(n)} = \frac{1}{N} \left( (\boldsymbol{vh})^{(1)} + (\boldsymbol{vh})^{(2)} + \cdots + (\boldsymbol{vh})^{(N)} \right) = \tag{3-39}$$

$$\frac{1}{N} \left[ \left( \begin{bmatrix} v_1 h_1 & \cdots & v_1 h_{500} \\ \vdots & \ddots & \vdots \\ v_{784} h_1 & \cdots & v_{784} h_{500} \end{bmatrix} \right)^{(1)} + \left( \begin{bmatrix} v_1 h_1 & \cdots & v_1 h_{500} \\ \vdots & \ddots & \vdots \\ v_{784} h_1 & \cdots & v_{784} h_{500} \end{bmatrix} \right)^{(2)} + \cdots + \left( \begin{bmatrix} v_1 h_1 & \cdots & v_1 h_{500} \\ \vdots & \ddots & \vdots \\ v_{784} h_1 & \cdots & v_{784} h_{500} \end{bmatrix} \right)^{(N)} \right]$$

All the observations are needed to estimate the expectation. This means we have to repeat the process of taking an image from MNIST, presenting to visible layer of positive machine, tracking the effects during the observation time ($T$), computing the probabilities

vectors and finally using (3-37) computing the desired matrix. This process can be done too fast in simulation for estimating the expectation by utilizing a parallel approach such as GPU or utilizing cluster computing. In this study we want to find suitable methods for using in a Neuromorphic Accelerator hardware. Also it is possible for this hardware approach to repeat the structures and create a parallel device, but this means more power and more hardware cost. Therefore in our simulation we divided the MNIST dataset to some mini-batches with *N* images [54]. In each mini-batch repeating the described process for *N* times and finally integrating the results and dividing by *N* (3-39), we have used a single structure (as described in 3.5.2.1.2). It's obvious this single implementation leads to less speed in computations.

Subsequently we have to do same computations in positive and negative machines to obtain $< \boldsymbol{vh} >_{data}$ and $< \boldsymbol{vh} >_{model}$. Eventually using the recent results we can compute the weight updating matrix:

$$\Delta \boldsymbol{w} = \eta(\mathbb{E}[\boldsymbol{vh}]_{data} - \mathbb{E}[\boldsymbol{vh}]_{model}) \qquad (3\text{-}40)$$

#### 3.5.2.1.5 The proposed algorithm

At this point, we have all the necessary tools to develop our learning model for our spike-based rate-coded RBM model.

***Divide*** *MNIST to* ***m*** *mini-batch with length* ***N***

***for each*** *mini-batch* ***do***

***{***

***for each*** *image* ***in*** *mini-batch* ***do***

***{***

***for each*** $\Delta t$ ***in*** $T$ ***d***

***{***

$\boldsymbol{s_v^+}$*=generated spikes vector (assuming pixel density as rate for Poisson process)*

$n_{s_v}^+ = n_{s_v}^+ + \boldsymbol{s_v^+}$

***Propagate*** $\boldsymbol{s_v^+}$

***Compute*** *input current for neurons in hidden layer of positive machine* (3-30)

***Update*** $\boldsymbol{s_h^+}$*(vector of neurons state in hidden layer of positive machine)* (3-29)

$n_{s_h}^+ = n_{s_h}^+ + \boldsymbol{s_h^+}$

***Propagate*** $\boldsymbol{s_h^+}$

***Compute*** *input current for neurons in visible layer of negative machine* (3-30)

***Update*** $\boldsymbol{s_v^-}$*(vector of neurons state in visible layer of negative machine)* (3-29)

$n_{s_v}^- = n_{s_v}^- + \boldsymbol{s_v^-}$

***Propagate*** $\boldsymbol{s_v^-}$

***Compute*** *input current for neurons in hidden layer of negative machine* (3-30)

***Update*** $\boldsymbol{s_h^-}$*(vector of neurons state in hidden layer of negative machine)* (3-29)

$n_{s_h}^- = n_{s_h}^- + \boldsymbol{s_h^-}$

*}*

***Update*** *probabilities vectors* (3-33)

***Compute***

$$\boldsymbol{vh_{data}} = \boldsymbol{P_v^+ P_h^+} = \begin{bmatrix} v_1h_1 & \cdots & v_1h_{500} \\ \vdots & \ddots & \vdots \\ v_{784}h_1 & \cdots & v_{784}h_{500} \end{bmatrix}$$

$$\boldsymbol{E_{neg_{temp}}} = \boldsymbol{E_{neg_{temp}}} + \boldsymbol{vh_{data}}$$

***Compute***

$$\boldsymbol{vh_{model}} = \boldsymbol{P_v^- P_h^-} = \begin{bmatrix} v_1h_1 & \cdots & v_1h_{500} \\ \vdots & \ddots & \vdots \\ v_{784}h_1 & \cdots & v_{784}h_{500} \end{bmatrix}$$

$$\boldsymbol{E_{pos_{temp}}} = \boldsymbol{E_{pos_{temp}}} + \boldsymbol{vh_{model}}$$

*}*

$$\boldsymbol{E_{pos}} = \frac{\boldsymbol{E_{pos_{temp}}}}{N}$$

$$\boldsymbol{E_{neg}} = \frac{\boldsymbol{E_{neg_{temp}}}}{N}$$

***Compute*** $\boldsymbol{\Delta w} = \eta(\boldsymbol{E_{pos}} - \boldsymbol{E_{neg}})$

***Update*** *weight matrix:* $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{\Delta w}$

*}*

#### 3.5.2.1.6 Spike-Based Rate-Coded Deep Belief Network

According to 2.4.3 developing a Deep Belief Network using Restricted Boltzmann machine is completely straightforward. Indeed utilizing the mentioned *greedy algorithm* (2.4.3) one can trains the RBMs and stacks them. In this study we first try to find the optimized parameters for the proposed RBM, and then we will proposed and test the stacked RBMs.

# Chapter 4:

## Method evaluation and results

# 4 Method evaluation and results

In this chapter we demonstrate the efficiency of the proposed method exploiting rate-coding aspect of spiking interactions for training Restricted Boltzmann Machine as the basic building block for deep architectures. Learning process in neural networks is related to many parameters (hyper-parameters [136]). For the sack of using Machine Learning methods and biological neural model, both in one model, we have too many adjustable parameters. Indeed finding the best configuration of the parameters for the proposed RBM, using global optimization algorithms such as *Genetic algorithm* is a good subject for future work. In this chapter practically and based on too many times model evaluating, and to have a base-point for evaluating the parameter's effects, we use a pre-defined configuration. In the following subsections keeping fixed all parameters except one, we try to find the best value for the changing parameter. We know changing one parameter may effect on other parameters and changing one parameter without changing the other ones is not always a right hypothesis. For example changing the membrane capacitance can changes many total properties of the model. Therefore we try to have a base model for all the observations and only changing less depended parameters, but actually it's a general approach to find approximately the best configuration of the parameters.

As was mentioned because of the hybrid property of our proposed model, the number of adjustment parameters are increased and the model has both of learning parameter of Restricted Boltzmann Machine training method and spiking neurons models. Consequently the parameters are divided to two main categories: *learning parameters* and *neuron parameters*.

Learning parameters have effect on the model since it's a RBM model and neuron parameters can change the results regarding the biological aspect of the used model of neurons. Following tables illustrate the most effective parameters. In [54, 136] one can find very helpful practical recommendations for parameters optimization.

Table 4-1 shows the *fixed parameters* during all of the trails. As it's illustrated all the parameters for neuron model postulated to be fixed. Also Table 4-2 shows the default values

of parameters for learning process.

*Table 4-1: Fixed parameters during all trials*

| **Fixed parameters** | | |
|---|---|---|
| **RBM Fixed Parameters** | | |
| **Parameters** | | **Value** |
| Number of hidden units | | 500 |
| Number of visible units | | 784+10 |
| $Gibbs_n$ (The number of Gibbs sampling process in each iteration) | | 1 |
| **Leaky Integrate-and-Fire fixed parameters** | | |
| **Parameters** | **Descriptions** | **Values** |
| $g_m$ | Membrane Conductance | 1 |
| $V_{rest}$ | Membrane rest voltage | 0 |
| $V_{reset}$ | Membrane reset voltage | 0 |
| $t_{refractory}$ | Refractory period of the LIF neurons | $2$ |
| $\Delta t$ | The required time for a vector of input spikes (during a time slot) to travel entire of model (from input in Positive Machine to output of Negative Machine) | $LIF_{SpikeProcceingTime} + Delay_{Syn_{v_{Pos}\to h_{Pos}}} + LIF_{SpikeProcceingTime} + Delay_{Syn_{h_{Pos}\to v_{Neg}}} + LIF_{SpikeProcceingTime} + Delay_{Syn_{v_{Neg}\to h_{Neg}}} + LIF_{SpikeProcceingTime}$ |
| $Delay_{Syn}$ | The required time for a given spike to pass from one neuron to another | 1ms |
| $LIF_{SpikeProcceingTime}$ | The needed time for processing the input spikes and generating the proper output for any given neurons | 0 |
| $\tau_{membrane}$ | LIF neurons membrane time constant | 1000ms |
| $V_{threshold}$ | Threshold voltage of the LIF neurons. In section 4.2.1 we describe it more. | $\begin{cases} 2 & neurons\ in\ visible\ layer \\ 0.5 & neurons\ in\ hidden\ layer \end{cases}$ |
| *Membrane voltage Noise* | Noisy input which can be studied as noise in membrane voltage | $\mathcal{N}(\mu = 0, \sigma^2 = 0.001)$ |
| *Threshold Noise* | Noise in output which allows the membrane potential to have an escape and generating a spike even in sub-threshold area. | $\mathcal{N}(\mu = 0, \sigma^2 = 0.01)$ |

*Table 4-2: Default values of Machine Learning parameters*

| **Parameters** | **Descriptions** | **Default Values** |
|---|---|---|
| $\eta$ | Learning rate | 0.005 |
| $Decay_w$ | Weight decay rate | 0.001 |
| $Initialize_w$ | Random matrix for initiating weight matrix | 0.01 |
| *Mini-batch-size* | The number of parallel training sample in each iteration | 50 |
| *Dataset size* | Number of images from MNIST training set which have been used for training the model | 20,000 |
| *Epochs* | Number of training iterations without resetting the weight matrix | 6 |
| *Max_Spk_Freq* | *Maximum Spike Frequency* | 3 |

## 4.1 Evaluation approach

The described method in [10] for evaluating Deep Belief Networks is explained in 2.4.3. In this chapter we train RBM first and based on the best result for the proposed RBM we develop the desired DBN.

The training set is a subset of MNIST ([106]). We choose 10,000 numbers on training set of MNIST such that it's consisted of equal number of each class of digits. Then using a uniform random process we extend the training set to 20,000 digits (some digits are repeated).

To find the optimized configuration of the RBM we need some ways for evaluating the proposed models. Here we describe two general approaches. The first one is a qualitative method and the second one is quantitative approach.

### 4.1.1 Monitoring learning process

As was mentioned in [54] using a graphical display, some of the common problems in learning process can be detected. Three types of display has been described in [54] for monitoring the reconstruction error. Using a graphical display of firing probability of hidden units as a gray scale image is described in [54] as a powerful tool. After each mini-batch observing this 2-dimensional matrix can be a measure of neurons activity in hidden layer. In fact it can illustrate if some neurons are never fired or if some are almost active during a

mini-batch. Because of the stochastic inherent of neural activates, if learning process works with no problem, no vertical or horizontal lines can be found in the graphical display.

Visualizing the connections between each hidden unit and all the visible units (visualizing weights) as a 28*28 image, for each hidden unit we can illustrate an image. The density of pixels in the corresponding images shows the connections strength. Then stronger connections are brighter and conversely about weaker connections. As will be shown in 4.2.1 a large amount of weights are negative then it's natural to see large dark regions in the images. Indeed each of this image can be thought as a filter which are applied on input images. When we want to compute the firing probability of a given hidden unit ($p(h_j = 1|\boldsymbol{v})$) in fact we are applying one of this filters on the given image vector. As is depicted in Figure 4-1 each image have some whiter lines or curves with some orientations, this means the corresponding hidden unit is sensitive to a given input image with same lines or curves. Then it's reasonable to describe an image as composition of this lines and curves. Also extremely when a networks is trained properly, it's possible for some specific hidden units to be sensitive to some specific class of input images (some specific numbers in MNIST). We can see such a described phenomenon in some thumbnail images in Figure 4-1.

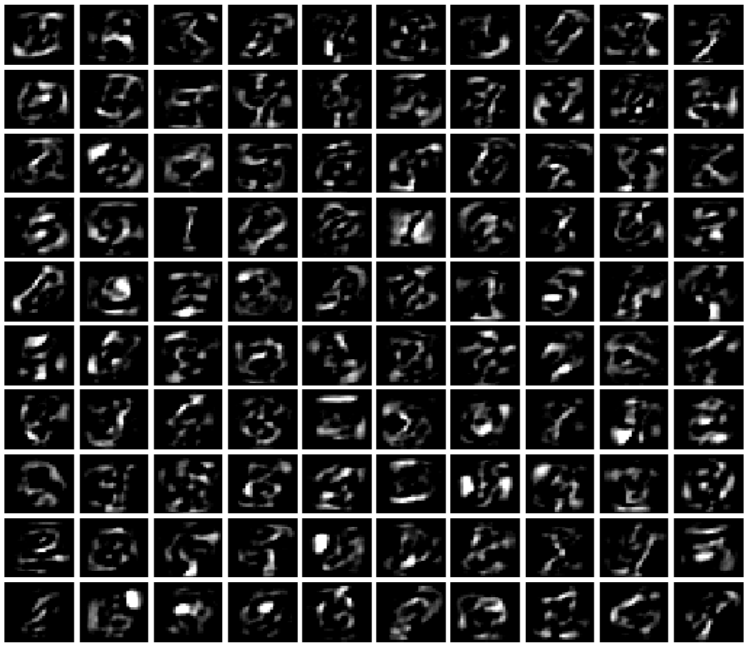

**Figure 4-1: Weights between each hidden and all visible units can be illustrated as a 28*28 images. The images can be interpreted as filters which make the corresponding hidden neurons sensitive to specific patterns of input vectors. The illustrated image is the weight matrix of a 784*100 RBM that has been trained using 60000 MNIST train images. Some depicted thumbnail images are very similar to some specific digits**

### 4.1.2 Energy Function

The described method for evaluation is not a numerical method. As was mentioned before the RBM learning rule is an unsupervised method. For evaluating the accuracy of the proposed model in classification tasks we need a proper approach. One of the helpful approaches to use RBMs for discrimination is to train the joint distribution of labels and data. The labels can be illustrated as *softmax* units. The softmax units represent the class of each label. In this study we postulate a 10 bits softmax vector to represent the class of labels so that all the units are zero except the corresponded bit of the given class. The input vector in composed of concatenating the describe vector as the label and the pixels vector of the given input image (Figure 4-2).

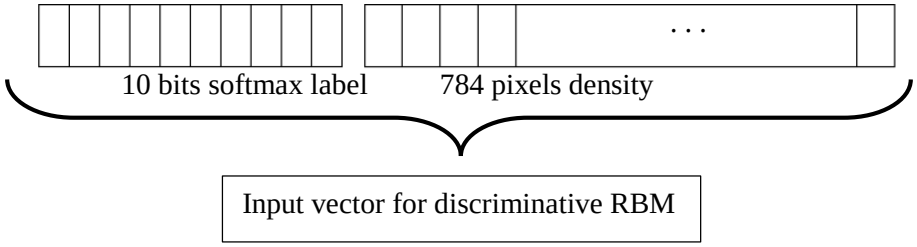


**Figure 4-2: Softmax training labels and training image vectors of MNSIT are concatenated as an input vector**

The proposed RBM model is depicted in Figure 4-3. After training, all the possible softmax labels with a given test vector will be evaluated via *Energy Function* (4-1) to find the lowest level of energy. Each label which can provide the lower energy is chosen as the detected class.

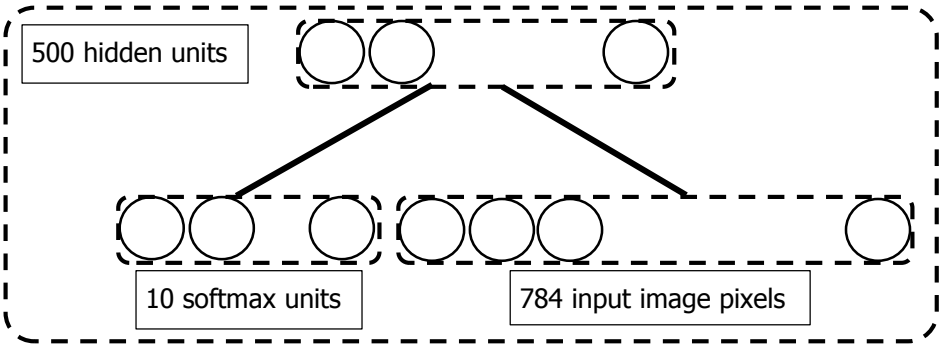


**Figure 4-3*: Discriminative RBM***

$$e^{-F(\boldsymbol{v})} = \Sigma_{\boldsymbol{h}} e^{E(\boldsymbol{v},\boldsymbol{h})}$$
$$where\ F(\boldsymbol{v}) = -\Sigma_i v_i a_i - \Sigma_j \log(1 + e^{x_j}) \quad (4\text{-}1)$$
$$and\ x_j = b_j + \Sigma_i v_i w_{ij}$$

## 4.2 Finding Optimized Parameters

Finding the optimized parameters for *Stochastic Gradient Descent (SGD)-Based* learning rules and specifically for RBMs, have been addressed in some literatures ([137, 138]).

In this subsection keeping fixed all parameters, we study the effect of changing only one parameter to find the best result. It is obvious better result for a given value of a specific parameter by keeping all other ones essentially does not mean the best configuration of the given parameter in all trials, indeed inconsistency between parameters is possible, but generally we are looking for a tradeoff between parameters.

### 4.2.1 Membrane voltage threshold adjustment

As presented in Table 4-1, we assume different threshold for units in hidden and visible layer. Regarding to the different number of units in visible and hidden layers, it is obvious units in hidden layers have more fan-in than visible ones. Then we predict the effect of weights on the hidden units are more intensive in average. Figure 4-4 illustrates the synaptic weights distribution. As we can see the negative weights population are more than the positive ones.

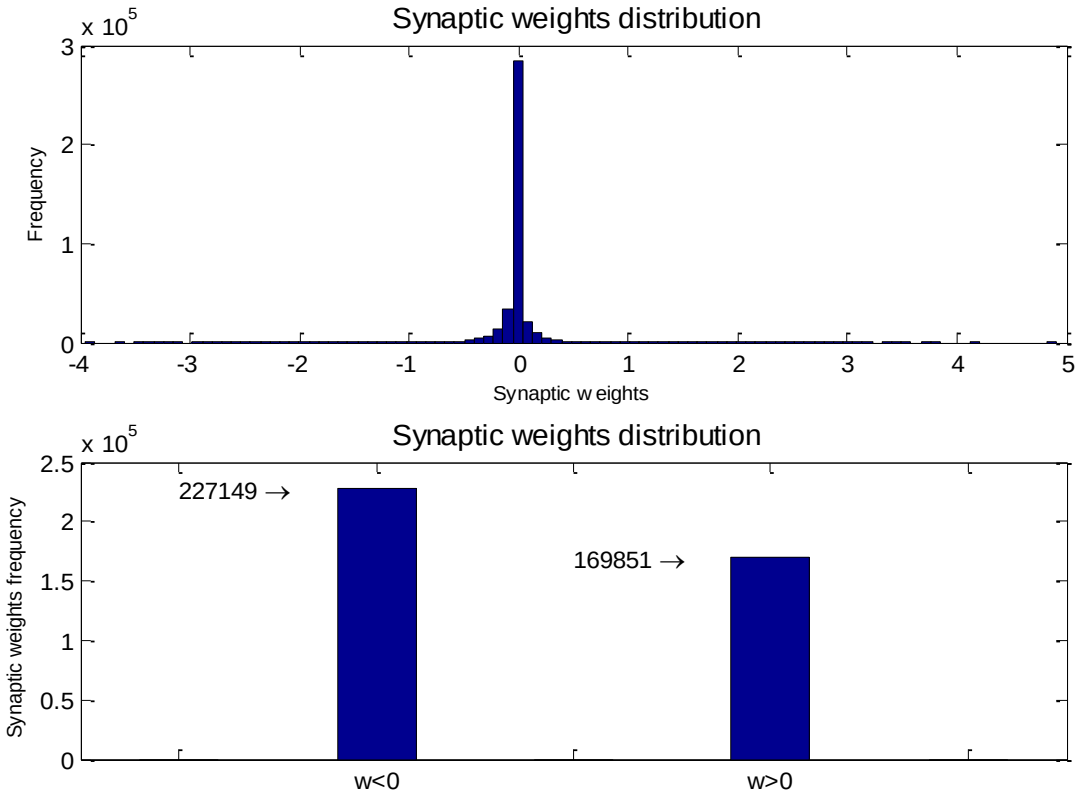


**Figure 4-4*: Synaptic weights distribution***

Then more fan-in for each units may leads to more negative input values. Therefore we predict the input value for each hidden unit in average, is less than the total input for each visible unit. To investigate this hypothesis we use a trained network and find the average of membrane potential of visible and hidden units in 1000 trails with 1000 different input spikes patterns. The results are depicted in Figure 4-5 and Figure 4-6 for hidden and visible units. In each observation the mean value of total input and membrane voltage of all units is computed. During 1000 trials we can see (Figure 4-5) the average of membrane potential for units in hidden layer, are more negative. Indeed for hidden units mean values membrane potential is between about -1 and -10 but the mean values of membrane potential for visible units are distributed between 0 and -3.5. Therefore it is reasonable to set two different threshold voltages for neurons in hidden and visible layers such that the threshold for neurons in visible layer is higher than the one for neurons in hidden layer. The values in Table 4-1 have been achieved according to some experiments and based on our assumptions.

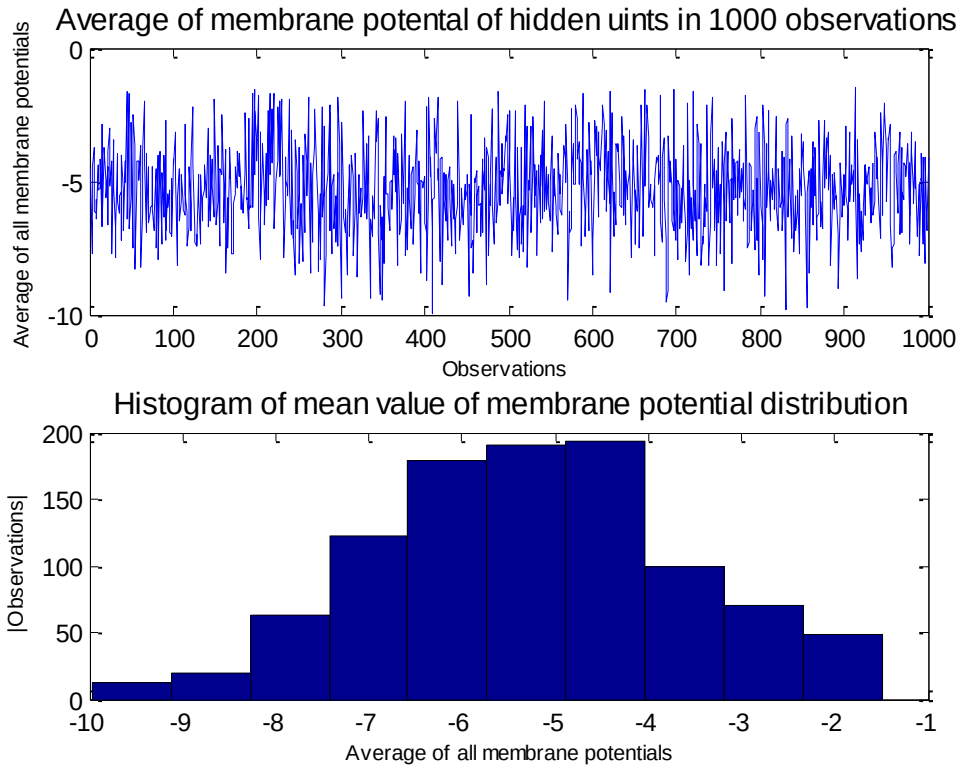


**Figure 4-5: Average of membrane potential of hidden units in 1000 observations**

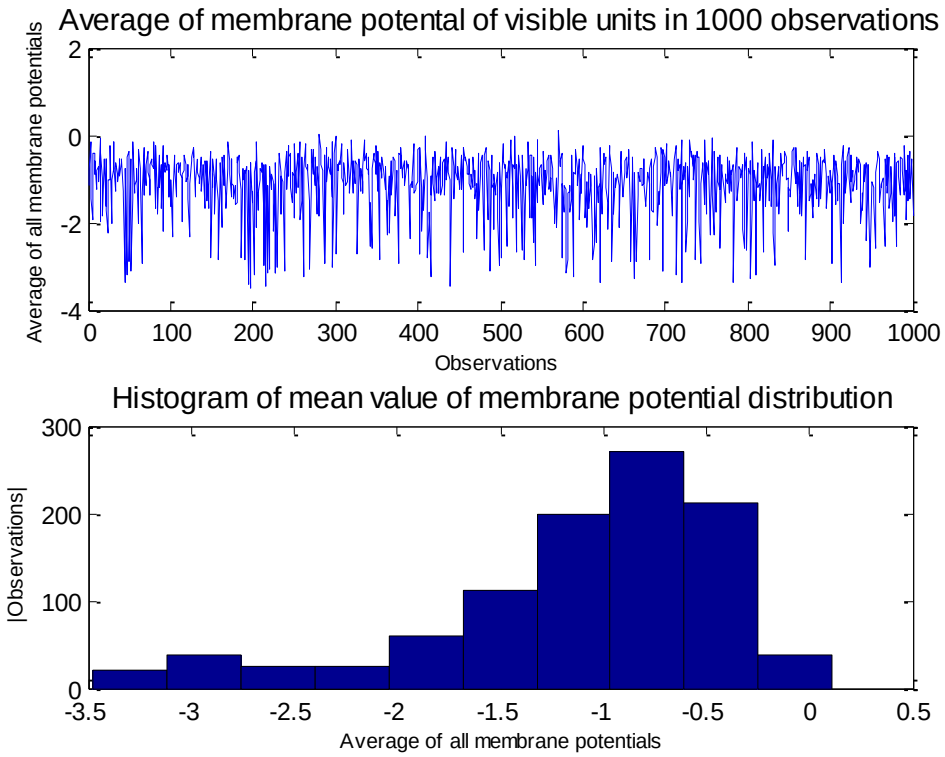


**Figure 4-6: Total input in average for visible units in 1000 observations**

### 4.2.2 Noise Effect

One of the well-known aspect of neural networks is their robustness dealing with noises. Indeed noise with regularizing the weights in spiking models can be useful and leads to generalization. Also respecting to *escape* phenomena (will be described below), it is possible for all neurons to have spikes. As was mentioned before in spiking models according to Hebbian learning rule, correlation between spike patterns in different layers of the model, that means correlation between pre- and post-synaptic neuron's activities, strengthens the synaptic connections and leads to learning. As depicted in Figure 4-7 when the network is not trained, neural activities in input layer have less (or no) effects on neural activities in post-synaptic layers. When the network is trained the visible layer in negative machine tries to imitate the neural activities of visible layer in positive machine and similarly the hidden layer in negative machine do same. As was discussed before Restricted Boltzmann Machine as a generative model tries to make close its internal representation to its environment, therefore as it is illustrated in Figure 4-8 during learning process and adjusting the weights the negative machine (as the reconstruction model of the positive machine), tries to represent the neural activities of the positive machine. Respecting to effect of spikes in learning process, it's important for all neurons to have spikes and noise can

improves the spike probability of neurons.

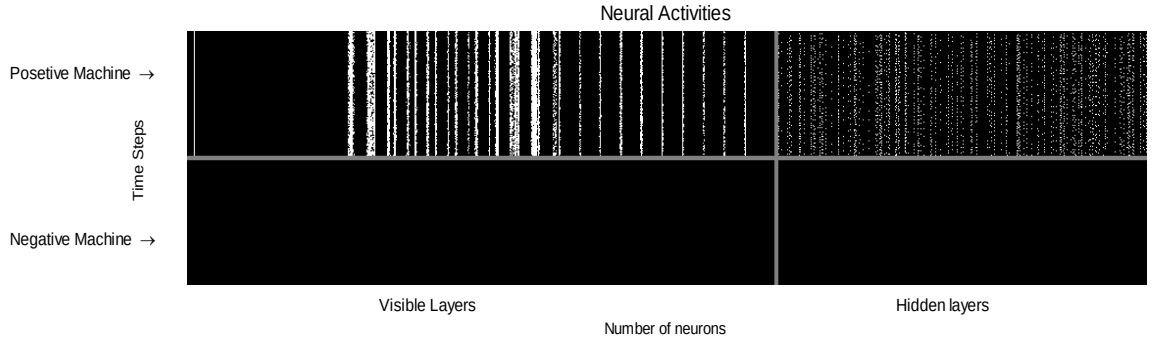


**Figure 4-7: Presenting spike stream of an image from MNIST to visible layer of Positive Machine when the weights are not adjusted. Here and for the sake of better illustration the presentation time is extended to $150\Delta t$. In each layer the top line is the first series of spike which are generated in first $\Delta t$.**

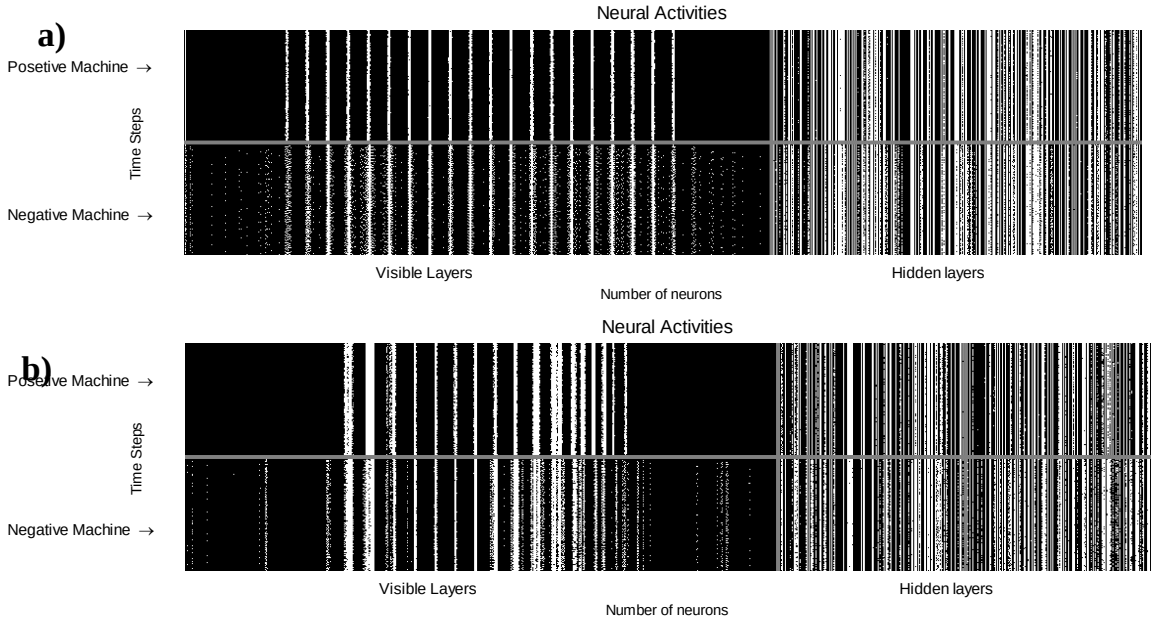


**Figure 4-8: Spike activities in a trained machine when relating to an input stream. a) Shows the spike correlation between visible-visible and hidden-hidden layers in Positive and Negative Machines with 75% accuracy. b) Shows the spike correlation between visible-visible and hidden-hidden layers in Positive and Negative Machines with 88% accuracy. As it's illustrated more spike activities correlation leads to higher accuracy.**

In [26] two main ways for introducing noise in spiking neuron models has been presented. Since in a biological system each post-synaptic neuron receives too many pre-synaptic inputs and considering the stochasticity of spike arrival times, the first effect of noise can be study as a noisy input current in the membrane voltage differential equation. Also this *noisy input* term can be alternatively studied as a noise in membrane voltages. Some types of noise (white noise and colored noise) has been explored in [26]. Here we use a simple Gaussian noise with mean 0 and variance 0.001 ($\mathcal{N}(\mu = 0, \sigma^2 = 0.001)$ to show the noise effect in membrane voltage. Concerning the intrinsic stochasticity of neurons it's possible for neurons to make a spike without reaching the membrane potential to threshold

voltage. In [26] the source of this *noisy output* is explained as a *soft threshold* which allows the membrane potential to have an *escape* and generating a spike even in sub-threshold area. Therefore we assume another Gaussian noise with mean 0 and variance 0.01 to have a noisy threshold (soft threshold). Figure 4-9 shows the effect of noise in membrane voltage when we added high level of noise to emphasis the effect of soft threshold.

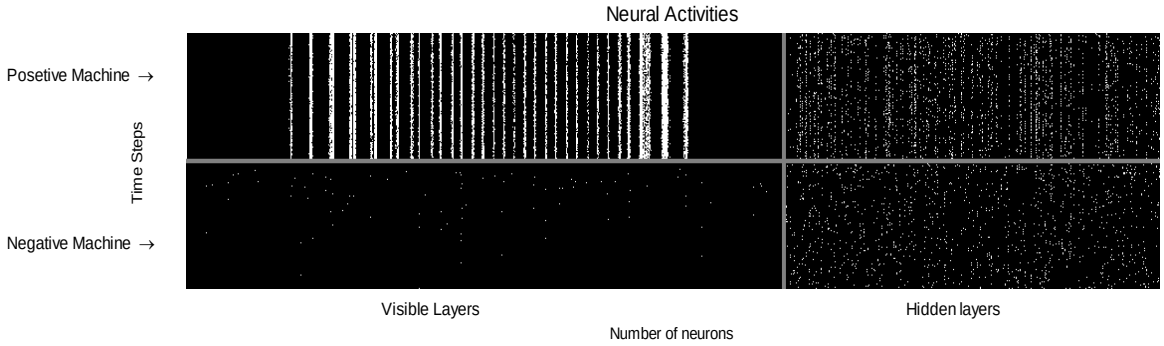


**Figure 4-9: Presenting spike stream of an image from MNIST to visible layer of the Positive Machine when the weights are not adjusted but we assume a high level of noise in threshold to show the escape and generating random spikes (Compare this image with Figure 4-7).**

### 4.2.3 Learning Rate

The learning rate is an important parameter in weights updating process. This parameter can changes the speed of convergence. Large learning rate cause to large weight updates and may leads to exploding the weights. Indeed in our model large learning rate causes to large value in weights and consequently cause to large change in membrane voltages that can leads to force a neuron to always send spikes or always be inactive. Also small learning rate usually causes to slow learning and it means we need more epochs to repeat the learning process and makes more time and more energy consuming which is not a good characteristic for a Neuromorphic model.

Here to investigate the effect of learning rate and finding the almost optimized value of learning rate, considering the recipes in [54] we compared some different learning rates in Figure 4-10. Also we know the best result in these experiments do not means the optimized value for learning rate. Indeed there is some better methods which can be used to optimized the results [57].

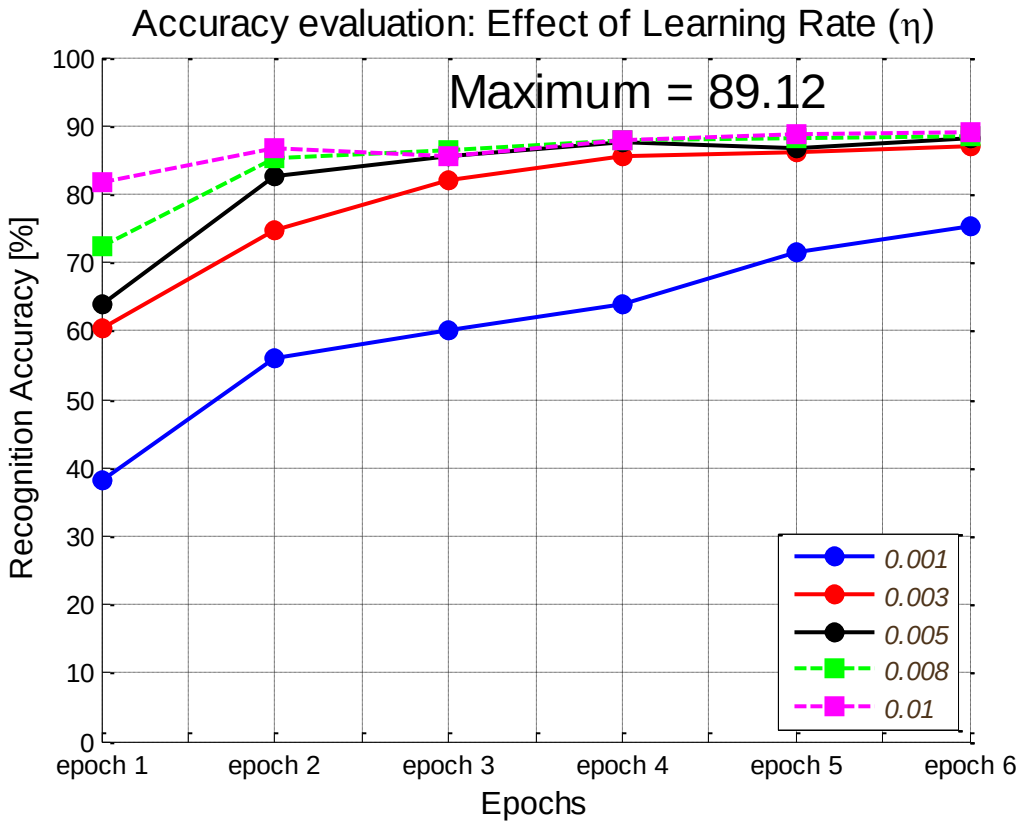


**Figure 4-10: Effect of learning rate in accuracy of the model: The model is evaluted with different learnig rates.**

### 4.2.4 Mini-batch size

Contrastive Divergence learning rule is a gradient based method. Indeed weight updating is possible after estimating the desired gradient. This estimation can be done pre each training data, but in order to decreasing the variance of estimated gradients and also for the sake of utilizing parallel computing and matrix operations opportunity, in [54] using mini-batches is suggested. In fact Contrastive Divergence is a *Stochastic Gradient Descent (GCD)* approximation algorithm. As was mentioned in [139] using stochastic gradient descent is a common method in machine learning tasks. In SGD iteratively a set of few examples of training set will be presented as input and the average gradient of the given examples will be computed to update the weights.

In this implementation since we are interested in hardware compatibility of our model (3.5.2.1.2 and 3.5.2.1.4), in each phase of mini-batch processing we don't use parallel methods to compute the effect of all input images at the same time. Because it means more costs, if one interested in boosting the speed of the proposed model, can does it by repeating the proposed architecture for the size of mini-batch. According to [54] for adjusting the size of each mini-batch we choose the default size of mini-batch as the number of classes in the

dataset (10). Also to decrease the error of sampling in gradient estimation for all the training data, each mini-batch must be consisted of all the classes. We compare the effect of the mini-batch size on recognition accuracy in Figure 4-11. The mini-batch size with best result is chosen for the size of mini-batches in DBN development.

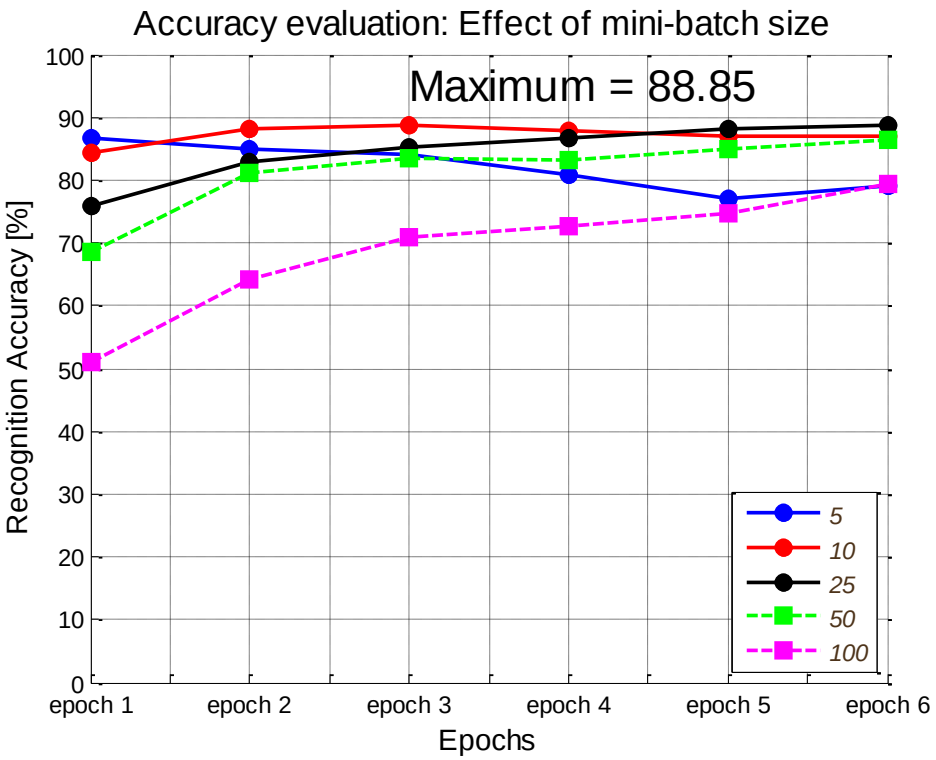


**Figure 4-11***: Effect of size of the mini-bathc in accuracy of the model.*

### 4.2.5 Spike Frequency

According to the described Poisson spike generation procedure, the maximum spike rate is depending on the maximum pixel density. It means we must assign the maximum number of spike in a given time to the pixel with maximum density. The number of spikes in a spike train and the inter spikes intervals are very important in learning process. Because of the leaky property of LIF neurons, for the enough long inter spikes intervals, according to the membrane time constant, the membrane potential will decreased and for a given neuron it's possible to has no spikes. Increasing the number of input spikes in a given time, can increases the firing probability of the post-synaptic units and also it can improve the spike correlation between pre- and post-synaptic neurons. As was mentioned in 3.5.2.1.2 in each $\Delta t$ we study entire of the network and track the input spikes from visible layer in positive machine to hidden layer in negative machine. Also we described this process can be repeated

for $k$ times such that $T = k\Delta t$. Each time slot ($\Delta t$) in our model is the total required time for the described spike propagation. Since we investigate the model from a multi-scale point of view, then it's possible to have different properties for each individual neurons. Here and for the sake of simplicity we assume zero neuron processing time for all neurons and also same synaptic delay for all links (1ms). Time increment in each neurons is not a factor of $\Delta t$. Indeed if $t_1$ is the arrival time of a given input spike then at $t_1 + Delay_{Syn}$ the effect of the input spike can reaches to the post-synaptic neurons. The firing probability for each post-synaptic neuron is a conditional and random event depending on its inputs which can occurred at each updating steps. In the proposed model each $\Delta t$ is consisted of $4Delay_{Syn}$. As it is illustrated in Figure 4-12 we assume an input synaptic delay for input spikes, this can be removed or can be attached to the output of the negative machine. According to our assumption the inputs are stopped until the output of the hidden layer in negative machine can be determined.

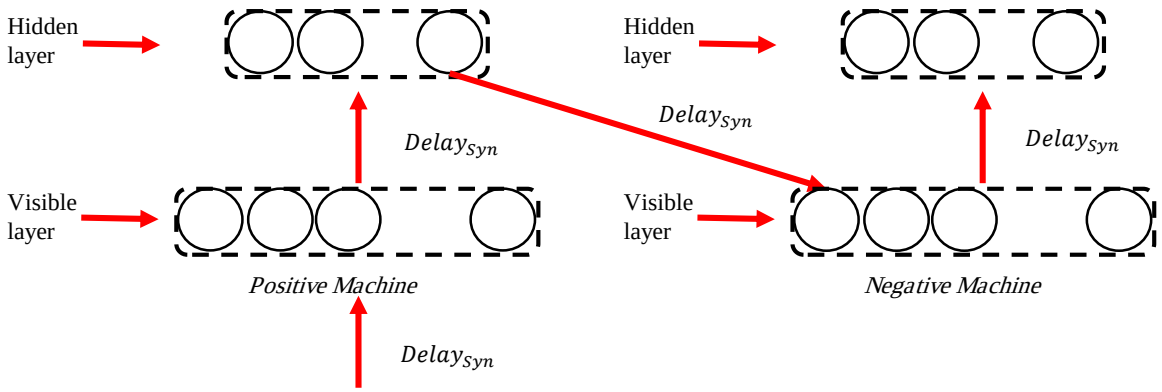


**Figure 4-12*: Delays in the proposed model***

Therefore one can imagines the minimum time of inter spikes interval is $4Delay_{Syn}$ and considering $Delay_{Syn} = 1ms$ the maximum frequency of the input spikes is:

$$spike\ frequency = \frac{1}{\Delta t} = \frac{1}{4ms} = 250\ Hz \qquad (4\text{-}2)$$

It means in one second we can fed the input layer with maximum 250 spikes and yet can track the spikes propagation in the network. Respecting to the updating algorithm (3.5.2.1.5) more input spikes means more updating time which is not desired. Therefore we test the model to find the minimum number of input spikes such that can train the network and leads to less time consuming. Indeed we are looking for the minimum number of

updating process iterations that can train the model. As depicted in Figure 4-13 we can see the effect of increasing the number of input spikes (the number of loops for updating and propagating spikes or the number of trials). Also regarding the Poisson spike generation process and assuming each updating as an independent experiment (and each $\Delta t$ as a time step) the input spike trains (considering each spike train as $k$ independent trials and each spike as a success and absence of spike as a fail), the Poisson distribution assumption is not broken.

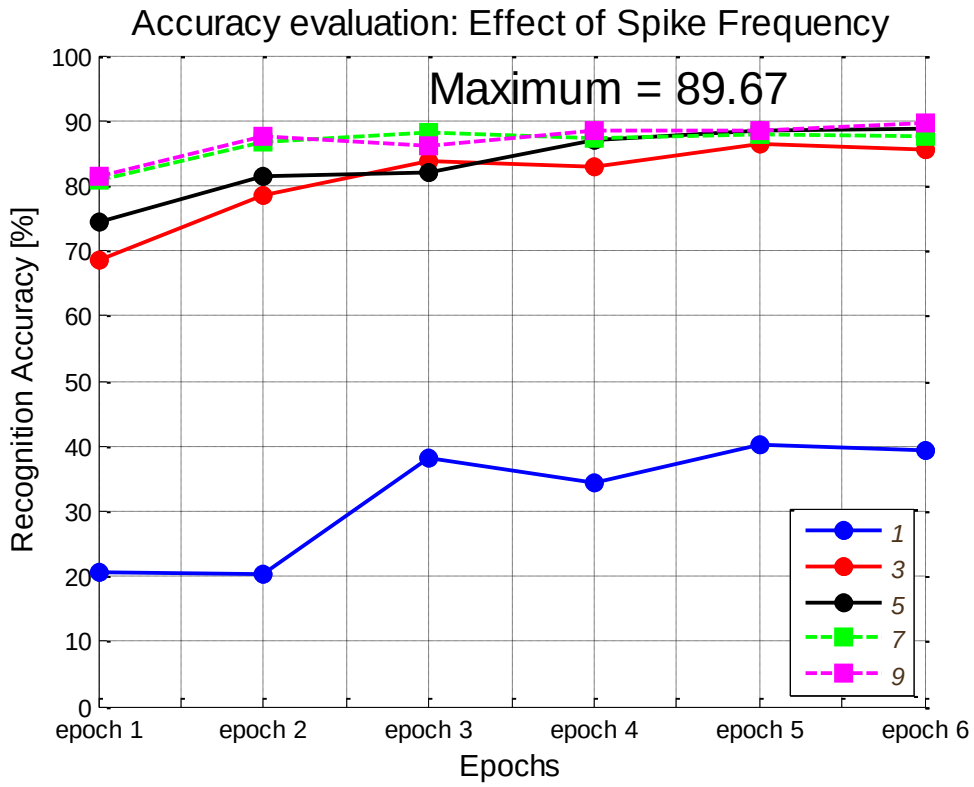


**Figure 4-13: Evaluating the effect of the number of spikes of any pixels during presantation time for any images, in the accuracy of the model.**

### 4.2.6 Optimized Configuration

Considering the above results in this subsection integrating all the corresponded values of parameters for the best results, Figure 4-15 illustrates the best achieved results for a single spiking RBM. In Figure 4-14 using confusion matrix the accuracy of the model in classifying the test digits is analyzed. High values along the main diagonal of confusion matrix indicates better classification and high values in other places means confusion in digits classification. In Figure 4-14 we used same number of images from each class of MNIST test set (800 images from each class). The spectrum shows the population of each square. Most of the squares are blue (has no members), which means no confusion in digits

classification. The most ambiguities are recorded for 4 which is detected as 9 (52 times), 7 which is classified as 9 (48 times), 3 which is identified as 8 (28 times) and 5 which is recognized as 3 (29 times). In the following we will develop a spike based DBN using RBMS with the final configuration (Table 4-3).

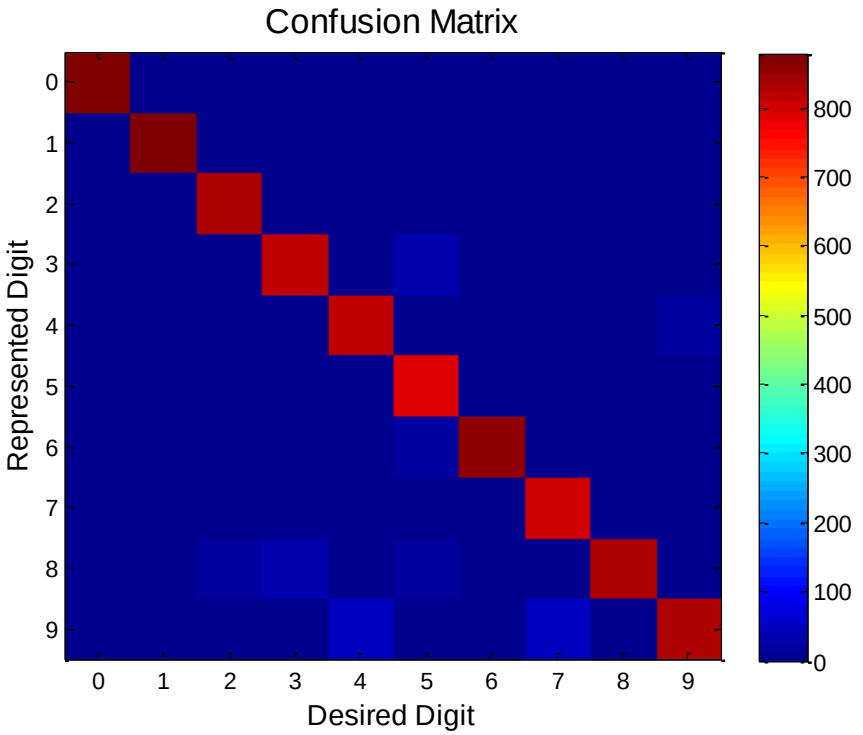


**Figure 4-14: The confusion matrix shows the confusion between two digits.**

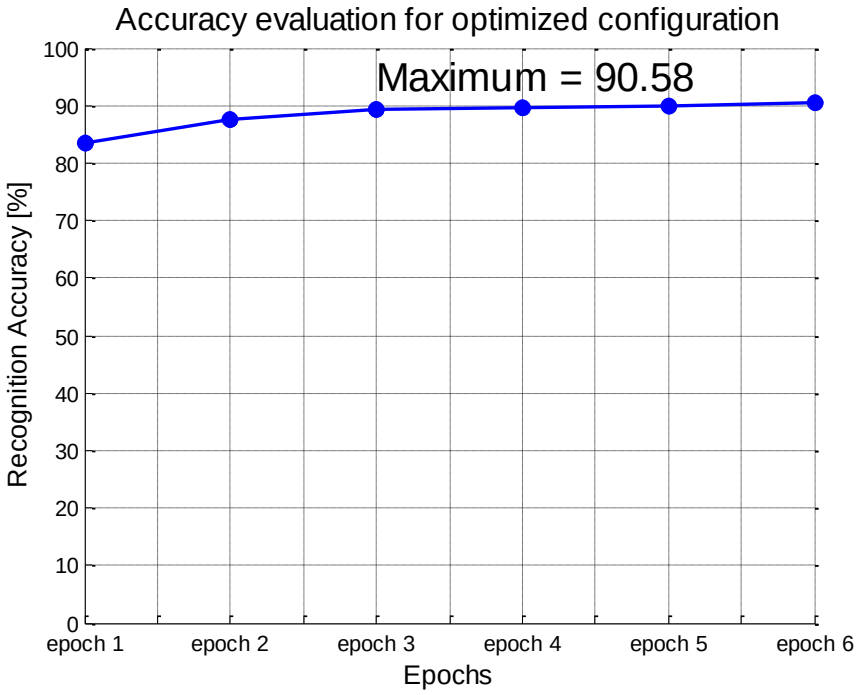


**Figure 4-15*: Evaluating optimized configuration***

**Table 4-3: The selected values for DBN development**

| Parameters | Descriptions | Default |
|---|---|---|
| $\eta$ | Learning rate | 0.01 |
| $Decay_w$ | Weight decay rate | 0.001 |
| $Initialize_w$ | Random matrix for initiating weight matrix | 0.01 |
| *Mini-batch-size* | The number of parallel training sample in each iteration | 25 |
| *Dataset size* | Number of images from MNIST training set which have been used for training the model | 20,000 |
| *Epochs* | Number of training iterations without resetting the weight matrix | 6 |
| *Maximum Number of spikes* | Maximum frequency of firing of a given neuron with maximum intensity | 9 |

## 4.3 The Proposed DBN

At this point we can develop a spike based Deep Belief Network by stacking the proposed spike-based Restricted Boltzmann Machines with the best configuration (Table 4-3) to find the best result for the proposed DBN (Figure 4-16). Figure 4-17 shows the plot of the obtained results. For evaluating this work, a list of some of the state-of-the-art work are presented and compared with the results of this paper in Table 4-4. Clearly the presented model not only provides an online model of spike-based DBN, but also has better results on MNIST handwritten digits recognition task. In this work only 20,000 images of MNIST training set have been used, where some mentioned work in

Table 4-4 had used extended version of MNIST.

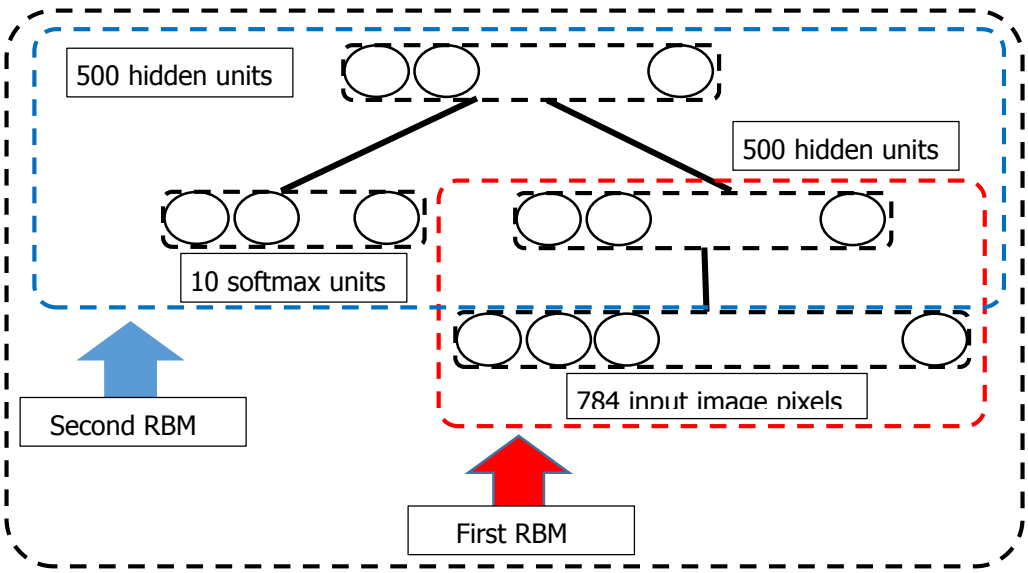

**Figure 4-16*: Stacking spike-based RBMs to build a spike-based DBN***

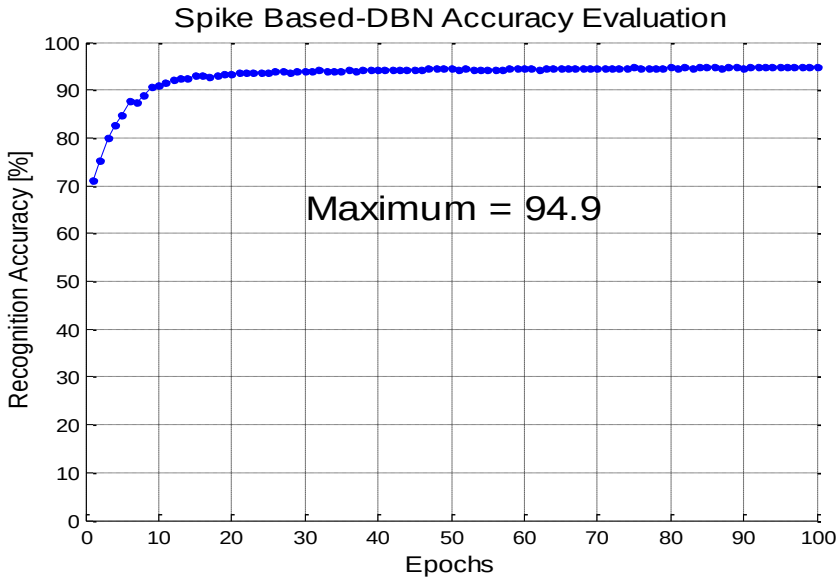


**Figure 4-17: Results of the proposed Spike-Based Deep Belief Network with best configuration (mini-batch size=25, spike frequency=9, learning rate=0.01)**

*Table 4-4: Comparing obtained results with some state-of-the-art results.*

| Introduced By | Architecture | The Neuron Types | Training Type | Learning-rule | Accuracy On MNIST (Free-Energy) |
|---|---|---|---|---|---|
| E.Neftci et al.2013 | 824*500 | LIF | Using Neural Sampling | STDP-Based CD | 90.8% |
| P.O'Connor et al. 2013 | 784*510*500 (DBN) | Siegert Neurons | Using offline method (transferring trained weights) | PCD | 94.1% |
| A.Muqeem Sheri et al.2015 | 824*500 | Stochastic Binary Units | Using two PCMO Memristors as a Synapse | CD | 87.85% |
| Our Proposed Method | 794*500 RBM | LIF | Online Training | CD | 90.58% |
| Our Proposed Method | (784*500)-(510*500) DBN | LIF | Online Training | CD | 94.9% |

# Chapter 5

## Conclusion and future works

# 5 Conclusion and Future Work

## 5.1 What We Proposed?

It is noticeable to clear that, our work is fundamentally distinguished from the proposed model by Y. W. Teh, and G. E. Hinton called RBMrate [140], because it did not use any types of biological spiking models and spikes had no roles, where our work proposed a RBM as a structure composed of LIF neurons and develop a DBN based on the proposed RBM.

According to subsection 3.2.1, the probability values in equations (3-33) are equal to the common definition of *mean firing rate* (3-12) when we do sampling for a while ($T$). From this point of view we can thought in the firing probability vectors as the mean firing rate vector of LIF neurons in each layer. Respecting to the Mean Field results for BM ((3-20) and (3-21)), intuitively one can find some resemblances. In fact it is not a far result, because Mean Field theory tries to analytically find the mean activities of an identical network from a macroscopic perspective and here we proposed numerically a procedure to find the mean firing rates in a non-identical network of LIF neurons. In an identical model of network, all the neurons are assumed to have same parameters such as same membrane time constant and same refractory time. But in our model each units can have its particular parameters. From this point of view the method is exploring the network at microscopic level.

Therefore we can conclude so that **"in the proposed model the sigmoidal probability functions in original Contrastive Divergence method (5-1), is replaced by common definition of *mean firing rate* of spiking neurons"**. Despite of CHL and original form of Contrastive Divergence, our model is a spiking model consisted of LIF neurons and not only its inputs are spike trains, but also the internal interactions between neurons are based on spikes. The neurons in the body of network, same as a biological tissue, receive spikes from pre-synaptic neurons and send spikes to post-synaptic neurons. Regarding some described process, each unit can computes its input firing rate base on the received spike trains from pre-synaptic neurons.

Then "***we proposed a spiked-based RBM which can learn through rate-coding***". In the other words we show using rate aspect of neural coding can be used efficiently to adjust the weights of Spike-Based Restricted Boltzmann Machines.

$$\Delta w_{ij} = < v_i h_j >_{data} - < v_i h_j >_{model}$$
$$\langle v_i h_j \rangle = m_i m_j \qquad (5\text{-}1)$$
$$where \;\; m_i = \sigma\left(\Sigma_j w_{ij} m_j + bais_i\right)$$
$$m_j = \sigma\left(\Sigma_i w_{ij} m_i + bais_j\right)$$

Using stochastic computations and sampling estimation, we explained the expectation of a set of observations. From this perspective we use the macroscopic aspect of neural sampling.

Therefore, we provided a link between microscopic and macroscopic aspect of neural sampling, this means "***the proposed model is a multi-scale model of neural activities***".

Considering biological aspect of neural networks and also using CD algorithm "***the model reduced the gap between Machine Learning domain and efficient neuronal computing aspect*** ".

Generally speaking, "***in this study using a multi-scale approach we proposed a model to close the gap between learning in RBM and neural activities and provide a spike-based model of RBM with LIF units such that can learn via rate-coding aspect of neural coding and also we develop a Deep Belief Network as a deep architecture built using the proposed spike-based RBM model***".

We train the model using 20,000 digits from MNIST in six epochs. The proposed model for a single RBM with 500 hidden units has 90.58% accuracy and 94.9% for a given DBN with 500 hidden units in the first RBM and also 500 units in the top layer. Since there is many adjustable parameters for both aspects of the model, then for boosting train process we just consider we considered the effect of a few parameters. Obviously more hidden units, deeper model, more epochs, using more input images and also using some machine learning tricks such as applying the effect of *momentum*, *inverse decay* and extending the input images using distortions may improve the results.

## 5.2 Future Work

In this dissertation we propose a framework for using RBM in an efficient model of spiking network. Therefore we can develop some idea using this framework. In the following we present a list of some applications we are going to implement using the presented model.

### 5.2.1 Using Memristor: Neuromorphic Implementation

Designing a circuit to keep the number of spikes for probability computing is a necessary step to implement a proper hardware for the proposed model. An alternative for the mentioned circuit is using a type of nano-devices which can keep the effects of spikes. It means we need a nano-device which its property(s) can change according to the number of incoming spikes. Our suggestion is using the Memristor nano-device. Indeed using the explained property of Memristor in [100] the model can consider the effects of the number of pulses. As is depicted in Figure 5-1, after each pulse the conductance of the nano-device will change according to the number of incoming pulses and as we know the Memristors are able to keep their last states in absence of power. The conductance can be tuned finely by the short voltage pulses. Respecting to the Hebbian and anti-Hebbian phases of Contrastive Divergence we can assign two types pulses for conductance potentiating and depression in corresponding phases to adjust the weights. We are looking for a way to fine tune this property of Memristors in a proper crossbar implementation. We hope to use Memristor in our future work.

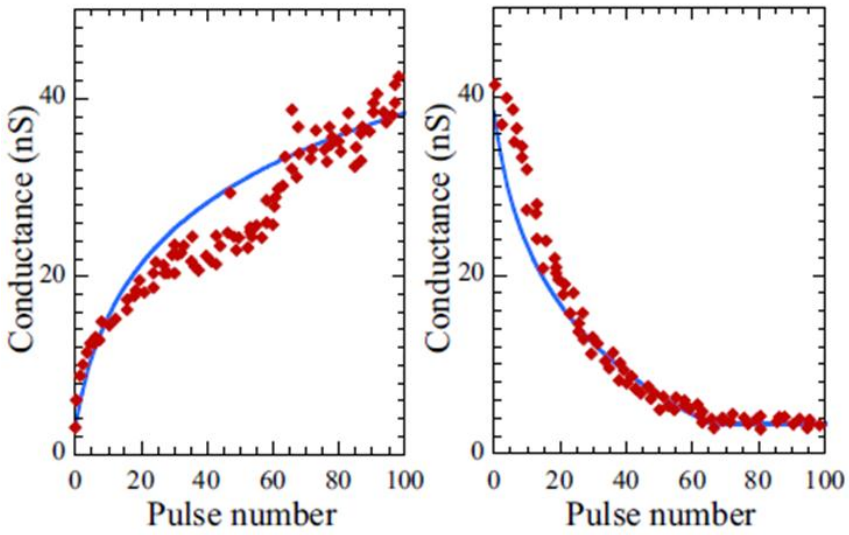


**Figure 5-1: Effect of potentiating and depressing pulses on Memristor conductance [100]**

### 5.2.2 Global Parameter Optimization

As was mentioned in previous chapter, there are too many parameters for fine tuning

the model. Proposing a proper cost function and considering the parameters as random variables, one can postulate a multi-variable model which can be optimized using heuristic solvers such as *Genetic* algorithm.

### 5.2.3 Parallel Implementation

For boosting the simulation speed, considering the time consuming process of expectation computation (3.5.2.1.4), we suggest utilizing a parallel approach such as GPU, FPGA or cloud computing.

### 5.2.4 Other Deep Architectures

There are some state-of-the-art deep models. Therefore we think it is possible to develop other deep structures utilizing methods same the proposed one in this work to develop other types of spiking deep models. *Convolutional Neural Networks (CNN)* as a bio-inspired and deep architecture has attracted many researchers in neural computing. Considering the good results of exploiting CNN in object recognition and image classification tasks [141, 142], text reading and language processing [143, 144] and voice and video processing [145-147], a same approach for using LIF neurons in Machine Learning domain can be utilized in CNN to develop a more realistic model of neural networks.

# Chapter 6
# Acknowledgements

# 6 Acknowledgements

I would like to thank Dr. Mahmood Ahmadi for his supervision, for his helpful suggestions and for his endless support and time. Also I thank Dr. Arash Ahmadi for his technical expertise and for helping me to find suitable papers and resources. My dear friend Mahyar Shahsavari suggested me good ideas for developing this thesis and indeed he helped me to be familiar with Neuromorphic computing. It was my pleasure to work with Professor Philippe Devienne and I thank him for his supports. A very special thanks to Dr. Jahanshah Kaboudian for his help to understanding and developing the mathematical methods in this work. Finally, this document is developed using the helpful suggestions of Professor Pierre Boulet and administrative supports of Computer Engineering Department of Razi University.

This research has been supported by the French-Iranian Gundishapour project and the Center of International Scientific Studies and Collaboration (CISS):"Neuromorphic Accelerators" Project 35855ZH, 2016-2017.

# References

# 7 References


[1] Y. Bengio, and Y. LeCun, "Scaling learning algorithms towards AI," *Large-scale kernel machines*, vol. 34, no. 5, 2007.

[2] Y. Bengio, "Learning deep architectures for AI," *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1-127, 2009.

[3] C. M. Bishop, *Pattern recognition and machine learning*, p.^pp. 44,226: springer New York, 2006.

[4] "Neuromorphic engineering," 2015/03/04; http://en.wikipedia.org/wiki/Neuromorphic_engineering.

[5] C. Mead, and M. Ismail, *Analog VLSI implementation of neural systems*: Springer Science & Business Media, 1989.

[6] "Human Brain Project," 2015/03/012; https://www.humanbrainproject.eu/.

[7] T. U. o. Manchester. "SpiNNaker Project " 2015/03/12; http://apt.cs.manchester.ac.uk/projects/SpiNNaker/project/.

[8] "SyNAPSE," 5/3/2016; https://en.wikipedia.org/wiki/SyNAPSE.

[9] J. De Villiers, and E. Barnard, "Backpropagation neural nets with one and two hidden layers," *Neural Networks, IEEE Transactions on*, vol. 4, no. 1, pp. 136-141, 1993.

[10] G. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural computation*, vol. 18, no. 7, pp. 1527-1554, 2006.

[11] "Google Brain," 5/3/2016; https://en.wikipedia.org/wiki/Google_Brain.

[12] R. D. Hof. "Deep Learning," April 15, 2015, 2015; http://www.technologyreview.com/featuredstory/513696/deep-learning/.

[13] D. B. Strukov, G. S. Snider, D. R. Stewart, and R. S. Williams, "The missing memristor found," *nature*, vol. 453, no. 7191, pp. 80-83, 2008.

[14] G. Indiveri, E. Chicca, and R. Douglas, "A VLSI array of low-power spiking neurons and bistable synapses with spike-timing dependent plasticity," *Neural Networks, IEEE Transactions on*, vol. 17, no. 1, pp. 211-221, 2006.

[15] B. Belhadj, A. Joubert, Z. Li, R. Héliot, and O. Temam, "Continuous real-world inputs can open up alternative accelerator designs." pp. 1-12.

[16] G. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural computation*, vol. 14, no. 8, pp. 1771-1800, 2002.

[17] T. M. Mitchell, "Machine learning. WCB," McGraw-Hill Boston, MA:, 1997, p. 2.

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, no. 3, pp. 1, 1988.

[19] W. S. McCulloch, and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115-133, 1943.

[20] M. Minsky, and P. Seymour, "Perceptrons," 1969.

[21] D. Kriesel, "A brief introduction to neural networks," *Retrieved August*, vol. 15, pp. 17, 2007.

[22] T. Serrano-Gotarredona, T. Masquelier, and B. Linares-Barranco, "Spike-Timing-Dependent-Plasticity with Memristors," *Memristor Networks*, pp. 211-247: Springer, 2014.

[23] "Chemical synapse," 2016/april/08; http://en.wikipedia.org/wiki/Chemical_synapse.

[24] W. Maass, "Networks of spiking neurons: the third generation of neural network models," *Neural networks*, vol. 10, no. 9, pp. 1659-1671, 1997.

[25] H. Paugam-Moisy, and S. Bohte, "Computing with spiking neuron networks," *Handbook of natural computing*, pp. 335-376: Springer, 2012.

[26] W. Gerstner, W. M. Kistler, R. Naud, and L. Paninski, *Neuronal dynamics: From single neurons to networks and models of cognition*: Cambridge University Press, 2014.

[27] L. Abbott, and W. Gerstner, "Homeostasis and learning through spike-timing dependent plasticity."

[28] W. Gerstner, and W. M. Kistler, *Spiking neuron models: Single neurons, populations, plasticity*: Cambridge university press, 2002.

[29] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio, "Why does unsupervised pre-training help deep learning?," *The Journal of Machine Learning Research*,

vol. 11, pp. 625-660, 2010.
[30] G. E. Hinton, and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504-507, 2006.
[31] N. Le Roux, N. Heess, J. Shotton, and J. Winn, "Learning a generative model of images by factoring appearance and shape," *Neural Computation*, vol. 23, no. 3, pp. 593-650, 2011.
[32] "Mathematical_model," April 02,2016; https://en.wikipedia.org/wiki/Mathematical_model.
[33] "Graphical model," 2/4/2016; http://en.wikipedia.org/wiki/Graphical_model.
[34] M. I. Jordan, and T. J. Sejnowski, *Graphical models: Foundations of neural computation*: MIT press, 2001.
[35] M. J. Wainwright, and M. I. Jordan, "Graphical models, exponential families, and variational inference," *Foundations and Trends® in Machine Learning*, vol. 1, no. 1-2, pp. 1-305, 2008.
[36] C. M. Bishop, and J. Lasserre, "Generative or discriminative? getting the best of both worlds," *Bayesian Statistics*, vol. 8, pp. 3-24, 2007.
[37] P. Smolensky, *Information processing in dynamical systems: Foundations of harmony theory*, DTIC Document, 1986.
[38] T. Schmah, G. E. Hinton, S. L. Small, S. Strother, and R. S. Zemel, "Generative versus discriminative training of RBMs for classification of fMRI images." pp. 1409-1416.
[39] J. J. Kivinen, and C. Williams, "Multiple texture Boltzmann machines." pp. 638-646.
[40] Y. Tang, R. Salakhutdinov, and G. Hinton, "Robust boltzmann machines for recognition and denoising." pp. 2264-2271.
[41] Q. V. Le, "Building high-level features using large scale unsupervised learning." pp. 8595-8598.
[42] A.-r. Mohamed, and G. Hinton, "Phone recognition using restricted boltzmann machines." pp. 4354-4357.
[43] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, and T. N. Sainath, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82-97, 2012.
[44] A.-r. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *Audio, Speech, and Language Processing, IEEE Transactions on*, vol. 20, no. 1, pp. 14-22, 2012.
[45] G. E. Hinton, and T. J. Sejnowski, "Learning and relearning in Boltzmann machines," *MIT Press, Cambridge, Mass*, vol. 1, pp. 282-317, 1986.
[46] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, pp. 2554-2558, 1982.
[47] A. Fischer, and C. Igel, "Training restricted Boltzmann machines: An introduction," *Pattern Recognition*, vol. 47, no. 1, pp. 25-39, 2014.
[48] J. Aldrich, "RA Fisher and the making of maximum likelihood 1912-1922," *Statistical Science*, vol. 12, no. 3, pp. 162-176, 1997.
[49] S. Kullback, and R. A. Leibler, "On information and sufficiency," *The annals of mathematical statistics*, vol. 22, no. 1, pp. 79-86, 1951.
[50] S. Geman, and D. Geman, "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, no. 6, pp. 721-741, 1984.
[51] K. Swersky, B. Chen, B. Marlin, and N. de Freitas, "A tutorial on stochastic approximation algorithms for training restricted Boltzmann machines and deep belief nets." pp. 1-10.
[52] J. R. A. C. Peterson, "A mean field theory learning algorithm for neural networks," *Complex systems*, vol. 1, pp. 995-1019, 1987.
[53] M. Welling, and G. E. Hinton, "A new learning algorithm for mean field Boltzmann machines," *Artificial Neural Networks—ICANN 2002*, pp. 351-357: Springer, 2002.
[54] G. E. Hinton, "A practical guide to training restricted boltzmann machines," *Neural Networks: Tricks of the Trade*, pp. 599-619: Springer, 2012.
[55] M. A. Carreira-Perpinan, and G. E. Hinton, "On contrastive divergence learning." pp. 33-40.
[56] T. Tieleman, "Training restricted Boltzmann machines using approximations to the

likelihood gradient." pp. 1064-1071.
[57] T. Tieleman, and G. Hinton, "Using fast weights to improve persistent contrastive divergence." pp. 1033-1040.
[58] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation." pp. 473-480.
[59] N. Le Roux, and Y. Bengio, "Representational power of restricted Boltzmann machines and deep belief networks," *Neural Computation*, vol. 20, no. 6, pp. 1631-1649, 2008.
[60] Y. LeCun, C. Cortes, and C. J. C. Burges. "THE MNIST DATABASE of handwritten digits," 2016/april/08; http://yann.lecun.com/exdb/mnist/.
[61] G. E. Dahl, T. N. Sainath, and G. E. Hinton, "Improving deep neural networks for LVCSR using rectified linear units and dropout." pp. 8609-8613.
[62] M. Gong, J. Zhao, J. Liu, Q. Miao, and L. Jiao, "Change detection in synthetic aperture radar images based on deep neural networks," *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 27, no. 1, pp. 125-138, 2016.
[63] X. Xiao, S. Zhao, D. H. H. Nguyen, X. Zhong, D. L. Jones, E. S. Chng, and H. Li, "Speech dereverberation for enhancement and recognition using dynamic features constrained deep neural networks and feature adaptation," *EURASIP Journal on Advances in Signal Processing*, vol. 2016, no. 1, pp. 1-18, 2016.
[64] R. Sarikaya, G. E. Hinton, and B. Ramabhadran, "Deep belief nets for natural language call-routing." pp. 5680-5683.
[65] V. Nair, and G. E. Hinton, "3D object recognition with deep belief nets." pp. 1339-1347.
[66] W. Diao, X. Sun, X. Zheng, F. Dou, H. Wang, and K. Fu, "Efficient Saliency-Based Object Detection in Remote Sensing Images Using Deep Belief Networks," 2016.
[67] G. Dahl, A.-r. Mohamed, and G. E. Hinton, "Phone recognition with the mean-covariance restricted Boltzmann machine." pp. 469-477.
[68] M. Jagadeesh, M. A. Kumar, and K. Soman, "Deep Belief Network Based Part-of-Speech Tagger for Telugu Language." pp. 75-84.
[69] G. W. Taylor, G. E. Hinton, and S. T. Roweis, "Modeling human motion using binary latent variables." pp. 1345-1352.
[70] S. Azizi, F. Imani, B. Zhuang, A. Tahmasebi, J. T. Kwak, S. Xu, N. Uniyal, B. Turkbey, P. Choyke, and P. Pinto, "Ultrasound-Based Detection of Prostate Cancer Using Automatic Feature Selection with Deep Belief Networks," *Medical Image Computing and Computer-Assisted Intervention--MICCAI 2015*, pp. 70-77: Springer, 2015.
[71] W. K. Mleczko, T. Kapuściński, and R. K. Nowicki, "Rough Deep Belief Network-Application to Incomplete Handwritten Digits Pattern Classification," *Information and Software Technologies*, pp. 400-411: Springer, 2015.
[72] "Deep Learning Research Groups," 2015/04/25; http://deeplearning.net/deep-learning-research-groups-and-labs/.
[73] A. L. Hodgkin, A. Huxley, and B. Katz, "Measurement of current-voltage relations in the membrane of the giant axon of Loligo," *The Journal of physiology*, vol. 116, no. 4, pp. 424-448, 1952.
[74] A. L. Hodgkin, and A. F. Huxley, "A quantitative description of membrane current and its application to conduction and excitation in nerve," *The Journal of physiology*, vol. 117, no. 4, pp. 500, 1952.
[75] A. L. Hodgkin, and A. F. Huxley, "The dual effect of membrane potential on sodium conductance in the giant axon of Loligo," *The Journal of physiology*, vol. 116, no. 4, pp. 497, 1952.
[76] A. L. Hodgkin, and A. F. Huxley, "Currents carried by sodium and potassium ions through the membrane of the giant axon of Loligo," *The Journal of physiology*, vol. 116, no. 4, pp. 449, 1952.
[77] P. Dayan, and L. F. Abbott, *Theoretical neuroscience*: Cambridge, MA: MIT Press, 2001.
[78] E. M. Izhikevich, "Simple model of spiking neurons," *IEEE Transactions on neural networks*, vol. 14, no. 6, pp. 1569-1572, 2003.
[79] E. M. Izhikevich, "Which model to use for cortical spiking neurons?," *IEEE transactions on neural networks*, vol. 15, no. 5, pp. 1063-1070, 2004.

[80] E. M. Izhikevich, and J. Moehlis, “Dynamical Systems in Neuroscience: The geometry of excitability and bursting,” *SIAM review*, vol. 50, no. 2, pp. 397, 2008.
[81] "Eugene M. Izhikevich old (NSI) home page," April 15,2016; http://www.izhikevich.org/.
[82] "Brain Corporation | Building brains for tomorrow's robots," April 15,2016; http://www.braincorporation.com/.
[83] D. Hebb, "0.(1949) The organization of behavior," Wiley, New York, 1968.
[84] W. Gerstner, R. Kempter, J. L. van Hemmen, and H. Wagner, “A neuronal learning rule for sub-millisecond temporal coding,” *Nature*, vol. 383, no. LCN-ARTICLE-1996-002, pp. 76-78, 1996.
[85] R. Kempter, W. Gerstner, and J. L. Van Hemmen, “Spike-based compared to rate-based Hebbian learning,” *Advances in neural information processing systems*, vol. 11, pp. 125-131, 1999.
[86] R. Kempter, W. Gerstner, and J. L. Van Hemmen, “Hebbian learning and spiking neurons,” *Physical Review E*, vol. 59, no. 4, pp. 4498, 1999.
[87] R. Kempter, W. Gerstner, and J. L. Van Hemmen, “Intrinsic stabilization of output rates by spike-based Hebbian learning,” *Neural Computation*, vol. 13, no. 12, pp. 2709-2741, 2001.
[88] W. Gerstner, and W. M. Kistler, “Mathematical formulations of Hebbian learning,” *Biological cybernetics*, vol. 87, no. 5-6, pp. 404-415, 2002.
[89] R. Brette, “Philosophy of the Spike: Rate-Based vs. Spike-Based Theories of the Brain,” *Frontiers in systems neuroscience*, vol. 9, 2015.
[90] D. H. Hubel, and T. N. Wiesel, “Receptive fields, binocular interaction and functional architecture in the cat's visual cortex,” *The Journal of physiology*, vol. 160, no. 1, pp. 106-154, 1962.
[91] W. Gerstner, “Population dynamics of spiking neurons: fast transients, asynchronous states, and locking,” *Neural computation*, vol. 12, no. 1, pp. 43-89, 2000.
[92] G.-q. Bi, and M.-m. Poo, “Synaptic modification by correlated activity: Hebb's postulate revisited,” *Annual review of neuroscience*, vol. 24, no. 1, pp. 139-166, 2001.
[93] P. Merolla, J. Arthur, F. Akopyan, N. Imam, R. Manohar, and D. S. Modha, "A digital neurosynaptic core using embedded crossbar memory with 45pJ per spike in 45nm." pp. 1-4.
[94] P. O'Connor, D. Neil, S.-C. Liu, T. Delbruck, and M. Pfeiffer, “Real-time classification and sensor fusion with a spiking deep belief network,” *Frontiers in neuroscience*, vol. 7, 2013.
[95] S. Hussain, S.-C. Liu, and A. Basu, "Improved margin multi-class classification using dendritic neurons with morphological learning." pp. 2640-2643.
[96] D. Neil, and S.-C. Liu, “Minitaur, an event-driven FPGA-based spiking network accelerator,” *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol. 22, no. 12, pp. 2621-2628, 2014.
[97] J. M. Brader, W. Senn, and S. Fusi, “Learning real-world stimuli in a neural network with spike-driven synaptic dynamics,” *Neural computation*, vol. 19, no. 11, pp. 2881-2912, 2007.
[98] S. Habenschuss, J. Bill, and B. Nessler, "Homeostatic plasticity in Bayesian spiking networks as Expectation Maximization with posterior constraints." pp. 773-781.
[99] M. Beyeler, N. D. Dutt, and J. L. Krichmar, “Categorization and decision-making in a neurobiologically plausible spiking network using a STDP-like learning rule,” *Neural Networks*, vol. 48, pp. 109-124, 2013.
[100] D. Querlioz, O. Bichler, P. Dollfus, and C. Gamrat, “Immunity to device variations in a spiking neural network with memristive nanodevices,” *Nanotechnology, IEEE Transactions on*, vol. 12, no. 3, pp. 288-295, 2013.
[101] P. U. Diehl, M. Cook, M. Tatsuno, and S. Song, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, 2015.
[102] B. Zhao, R. Ding, S. Chen, B. Linares-Barranco, and H. Tang, “Feedforward categorization on AER motion events using cortex-like features in a spiking neural network,” *Neural Networks and Learning Systems, IEEE Transactions on*, vol. 26, no. 9, pp. 1963-1978, 2015.
[103] E. Stromatias, D. Neil, F. Galluppi, M. Pfeiffer, S.-C. Liu, and S. Furber, “Live demonstration: handwritten digit recognition using spiking deep belief networks on SpiNNaker,” 2015.

[104] G. Indiveri, B. Linares-Barranco, and T. Serrano-Gotarredona, "Neuromorphic silicon neuron circuits," 2011.
[105] J. Borghetti, Z. Li, J. Straznicky, X. Li, D. A. Ohlberg, W. Wu, D. R. Stewart, and R. S. Williams, "A hybrid nanomemristor/transistor logic circuit capable of self-programming," *Proceedings of the National Academy of Sciences,* vol. 106, no. 6, pp. 1699-1703, 2009.
[106] E. Neftci, S. Das, B. Pedroni, K. Kreutz-Delgado, and G. Cauwenberghs, "Event-driven contrastive divergence for spiking neuromorphic systems," 2013.
[107] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons," *PLoS Comput Biol,* vol. 7, no. 11, pp. e1002211, 2011.
[108] J. Fiser, P. Berkes, G. Orbán, and M. Lengyel, "Statistically optimal perception and learning: from behavior to neural representations," *Trends in cognitive sciences,* vol. 14, no. 3, pp. 119-130, 2010.
[109] A. J. Siegert, "On the first passage time probability problem," *Physical Review,* vol. 81, no. 4, pp. 617, 1951.
[110] F. Jug, J. Lengler, C. Krautz, and A. Steger, "Spiking networks and their rate-based equivalents: does it make sense to use Siegert neurons?," *Swiss Society for Neuroscience,* 2012.
[111] S. Ostojic, and N. Brunel, "From spiking neuron models to linear-nonlinear models," *PLoS Comput Biol,* vol. 7, no. 1, pp. e1001056, 2011.
[112] E. P. Simoncelli, L. Paninski, J. Pillow, and O. Schwartz, "Characterization of neural responses with stochastic stimuli," *The cognitive neurosciences,* vol. 3, pp. 327-338, 2004.
[113] H. R. Wilson, and J. D. Cowan, "Excitatory and inhibitory interactions in localized populations of model neurons," *Biophysical journal,* vol. 12, no. 1, pp. 1, 1972.
[114] H. R. Wilson, and J. D. Cowan, "A mathematical theory of the functional dynamics of cortical and thalamic nervous tissue," *Kybernetik,* vol. 13, no. 2, pp. 55-80, 1973.
[115] P. L. Nunez, "The brain wave equation: a model for the EEG," *Mathematical Biosciences,* vol. 21, no. 3, pp. 279-297, 1974.
[116] S.-i. Amari, "Dynamics of pattern formation in lateral-inhibition type neural fields," *Biological cybernetics,* vol. 27, no. 2, pp. 77-87, 1977.
[117] A. Destexhe, and T. J. Sejnowski, "The Wilson–Cowan model, 36 years later," *Biological cybernetics,* vol. 101, no. 1, pp. 1-2, 2009.
[118] S. Coombes, P. beim Graben, R. Potthast, and J. Wright, *Neural Fields*: Springer, 2014.
[119] T. S. Lee, and D. Mumford, "Hierarchical Bayesian inference in the visual cortex," *JOSA A,* vol. 20, no. 7, pp. 1434-1448, 2003.
[120] D. P. Reichert, P. Series, and A. J. Storkey, "A hierarchical generative model of recurrent object-based attention in the visual cortex," *Artificial Neural Networks and Machine Learning–ICANN 2011*, pp. 18-25: Springer, 2011.
[121] D. P. Reichert, P. Series, and A. J. Storkey, "Charles Bonnet syndrome: evidence for a generative model in the cortex?," *PLoS Comput Biol,* vol. 9, no. 7, pp. e1003134, 2013.
[122] I. Stoianov, M. Zorzi, S. Becker, and C. Umilta, "Associative arithmetic with Boltzmann Machines: The role of number representations," *Artificial Neural Networks—ICANN 2002*, pp. 277-283: Springer, 2002.
[123] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines*," *Cognitive science,* vol. 9, no. 1, pp. 147-169, 1985.
[124] G. E. Hinton, "Deterministic Boltzmann learning performs steepest descent in weight-space," *Neural computation,* vol. 1, no. 1, pp. 143-150, 1989.
[125] J. R. Movellan, "Contrastive Hebbian learning in the continuous Hopfield model." pp. 10-17.
[126] A. D. Brown, "Spiking boltzmann machines." p. 122.
[127] Y. LeCun, and C. Cortes, "MNIST handwritten digit database," *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2010.
[128] M. Fatahi, *MNIST handwritten digits*, https://www.researchgate.net/publication/273124795_MNIST_handwritten_digits_Description_and_using, 2014.

[129] D. Querlioz, W. S. Zhao, P. Dollfus, J.-O. Klein, O. Bichler, and C. Gamrat, "Bioinspired networks with nanoscale memristive devices that combine the unsupervised and supervised learning approaches." pp. 203-210.
[130] G. Indiveri, F. Corradi, and N. Qiao, "Neuromorphic Architectures for Spiking Deep Neural Networks," 2015, 2015.
[131] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, "A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128K synapses," *Frontiers in neuroscience,* vol. 9, 2015.
[132] P. Lichtsteiner, C. Posch, and T. Delbruck, "A 128× 128 120 dB 15 μs latency asynchronous temporal contrast vision sensor," *Solid-State Circuits, IEEE Journal of,* vol. 43, no. 2, pp. 566-576, 2008.
[133] D. Heeger, "Poisson model of spike generation," *Handout, University of Standford,* vol. 5, 2000.
[134] R. E. Kass, and V. Ventura, "A spike-train probability model," *Neural computation,* vol. 13, no. 8, pp. 1713-1720, 2001.
[135] L. Buesing, J. Bill, B. Nessler, and W. Maass, "Neural dynamics as sampling: a model for stochastic computation in recurrent networks of spiking neurons," *PLoS computational biology,* vol. 7, no. 11, pp. e1002211, 2011.
[136] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," *Neural Networks: Tricks of the Trade*, pp. 437-478: Springer, 2012.
[137] K. Cho, T. Raiko, and A. T. Ihler, "Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines." pp. 105-112.
[138] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng, "On optimization methods for deep learning." pp. 265-272.
[139] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 7553, pp. 436-444, 2015.
[140] Y. W. Teh, and G. E. Hinton, "Rate-coded restricted Boltzmann machines for face recognition," *Advances in neural information processing systems,* pp. 908-914, 2001.
[141] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks." pp. 1097-1105.
[142] L. A. Alexandre, "3D object recognition using convolutional neural networks with transfer learning between input channels," *Intelligent Autonomous Systems 13*, pp. 889-898: Springer, 2016.
[143] B. Hu, Z. Lu, H. Li, and Q. Chen, "Convolutional neural network architectures for matching natural language sentences." pp. 2042-2050.
[144] M. Jaderberg, K. Simonyan, A. Vedaldi, and A. Zisserman, "Reading text in the wild with convolutional neural networks," *International Journal of Computer Vision,* vol. 116, no. 1, pp. 1-20, 2016.
[145] O. Abdel-Hamid, A.-r. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *Audio, Speech, and Language Processing, IEEE/ACM Transactions on,* vol. 22, no. 10, pp. 1533-1545, 2014.
[146] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks." pp. 1725-1732.
[147] K. Kang, W. Ouyang, H. Li, and X. Wang, "Object detection from video tubelets with convolutional neural networks," *arXiv preprint arXiv:1604.04053,* 2016.