The Next Generation Neural Networks: Deep Learning and Spiking Neural Networks

Erdem Başeğmez

NEUROSCIENTIFIC SYSTEM THEORY
Technische Universität München

27.06.2014

Abstract

Deep Learning and Spike Neural Networks are hot topics in artificial intelligence and human brain. By explaining the basic underlying blocks beneath them, the architectures and applications of both concepts are discovered.

Contents

Chapter 1

Introduction to Neural Networks

How the brain learns is one of the unsolved questions in science [RCO00]. The nervous system is a composition of neurons and glia. Neurons transmit impulses in the nervous system [Kal09]. Neurons, which are the base elements of the brain, are introduced using models in this section.
The neuron is the basic block, which processes information in the sense of cognition. Scientists mostly prefer to use simplified models of a neuron in order to decrease computational costs [RCO00]. A neuron consists of nucleus, dendrites and axons. The dendrites are the inputs to the system and the outputs of this system are axons. The axons and dendrites are connected to each other via synapses.

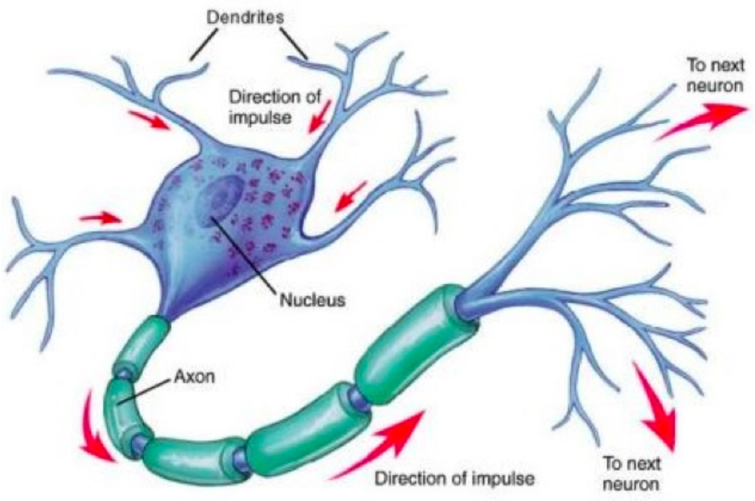


Figure 1.1: A neuron model [And95]

Theoretically, a postsynaptic neuron is connected to presynaptic neurons via synapses and these connections have different weights. The postsynaptic neuron gets activated and reacts by creating a spike when the incoming spikes have reached a threshold level. It is assumed that the neuron's state is on when it fires a spike and off when it does not fire a spike. This is the basic model being used for neural networks. A model for the activation function of a neural network is described as:

$$Y = f(b + \sum(x_i * w_i)) \tag{1.1}$$

where b is the bias, x is the input and w is the weight of the input. $f(.)$ is a nonlinear function and y is the output. According to this model inputs and outputs are nonlinear functions and outputs are at on state when a threshold is reached [And95].
Neural networks are especially useful for machine learning problems. The general purpose of machine learning can be explained with an easy example. Assuming that there are images, which may or may not include ducks. The expectation is to let the machine figure out which images have the ducks and which do not. In order to do that, the machine is first fed with training images, where it learns the features. After the learning phase, it is expected from the machine to give correct results when it is fed with new images.
Neuron models can be grouped into three generations:

1. Neurons with threshold activation functions: Perceptrons

2. Neurons with continous activation functions

3. Spiking neurons

1.1 First Generation Neurons: Perceptron

Perceptrons are the basic neural network building blocks, which are used for training algorithms. The procedure of classification using perceptrons can be explained again with an example. Assuming there are labeled samples in the plane, which should be separated into two groups. An easy way of separation is to draw a line between these groups. In this case, the separator line would be a linear classifier. The samples above the line would make one group and below ones the other group.

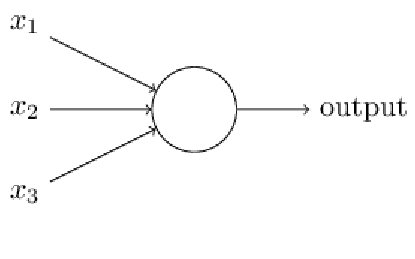


Figure 1.2: Perceptron model, x are inputs [nie]

The linear classifier here can be represented mathematically with:

$$F(x) = xw + b \tag{1.2}$$

where x is the input vector, w is the vector of weights and b is the bias.
The activation function $H(x)$ of a perceptron, which produces the result, can be described as follows:

$$H(x) = 1 \text{ if } f(x) > 0, \text{ otherwise } 0 \tag{1.3}$$

In order to train a perceptron, it is first fed by multiple training samples and the perceptron calculates the output. During the training phase the w vector is adjusted in order to minimize the output error, which is basically the difference between the output of the perceptron and the desired result.
A major problem with perceptrons is that their activation functions produce binary value only. During learning, a small change in the weight can cause a flip of the output, which may affect the behavior of the system. The weights should be adjusted gradually to reach the desired behavior, which is possible using different activation functions. This will make it possible for the new neurons to produce values between 0 and 1. In short, the problem of the perceptrons is that they can only realize linearly separable functions [top, nie].

1.2 Architectures of Neural Networks

There are several architectures of neural networks, such as single-layer perceptron, multilayer perceptron, competitive networks, self-organizing map, recurrent networks etc. [Sam07].
Popular neural networks are feedforward and recurrent neural networks. In the feedforward neural networks, the connection between the input and output is unidirectional and there is no connection between the neurons in a layer. The layer between the input and output layer is called hidden layer. The network is called deep neural network, if there is more than one hidden layer. Across the layers the similarity of a property may increase while another's decreases. This condition requires the definition of the layers by non-linear functions. By definition, the feedforward network consists of an input layer, an output layer and hidden layers, whose neurons are connected to each other bottom-up across the layers with directional connections and weights. Each neuron in the hidden layers calculates its output using the activation function, which is a function of its weighted sums. Using this procedure the values are propagated from the input layer through hidden layers to the output layer. Considering second generation neurons, the activation function may be different than $H(x)$.
In recurrent neural networks (RNN) there may be connections between the neurons in a layer. RNNs are more realistic in a biological manner but are harder to train

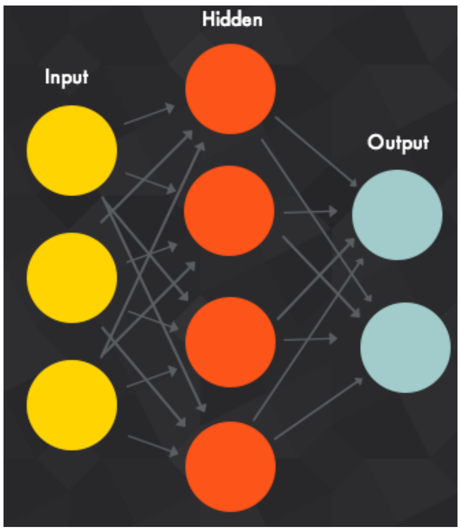


Figure 1.3: Feedforward neural network [top]

due to higher complexity. Recurrent networks make it possible to remember the context of the sequential data because of their structure.
For some applications, it is crucial to consider the past and future elements of a sequence.
The purpose of building neural networks is to learn information from the data input [Sam07]. The neural networks can be used for feature extraction, prediction, data classification etc.

1.3 Learning in Neural Networks

Learning in the neural networks basically means adjustment of the weights of the connections between the neurons in the system. Various learning algorithms in the neural networks are based on the Hebbian learning rule. According to the Hebbian, if two neurons are active simultaneously, the weight of the connection between them should be increased.
Different architectures use different learning algorithms. Commonly involved algorithms are gradient descent, backpropagation and contrastive divergence.

1.3.1 Training with gradient descent

A network is expected to provide the minimum output error. The output error $C(w, b)$ can be defined as:

$$C(w, b) = 0.5 * \sum(\|y(x) - a\|)^2 \tag{1.4}$$

where x is the input, y is the desired output and a is the output of the network.

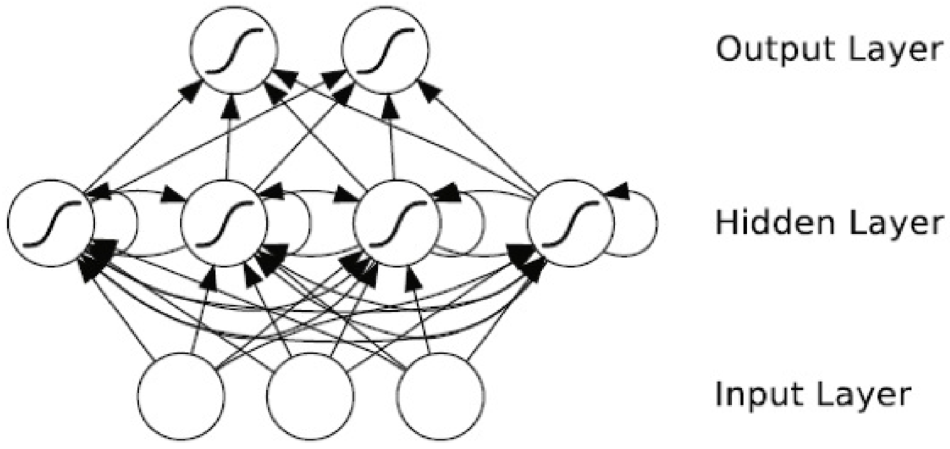


Figure 1.4: Recurrent neural network [Gra12a]

The output a of a neuron is the result of the action function, which depends on the weights of the connection and the bias. Shortly, the purpose of the training phase is to minimize $C(w, b)$.
Using calculus to solve the minimization problem will be difficult for networks with high amount of neurons; so gradient descent algorithm is preferred to obtain the optimum weights.

$$w_{new} = w_{old} - \gamma \frac{\partial C}{\partial w} \tag{1.5}$$

$$b_{new} = b_{old} - \gamma \frac{\partial C}{\partial b} \tag{1.6}$$

where γ is the learning rate.
The goal of these rules is to move in the direction of the gradient. The derivate of the output error is hard to calculate for each training sample. Thus stochastic gradient descent method is preferred to speed up the training phase. Stochastic gradient descent method estimates the true gradient by taking small amount of samples and averaging them [nie, top].

1.3.2 Backpropagation

Backpropagation, which is a prominent algorithm for computing the gradient, is used to calculate the gradients. The procedure of backpropagation can be explained as follows:
The network is given the input and the neurons are activated until the highest layer. Using the equations of backpropagation the output error of the highest layer is calculated. The error is backpropagated, which means the error of the former layer is found. According to these equations, the gradient of the error function is indeed the multiplication of the output error of the neuron and the activation

function of the stimulating neuron in the former layer. This procedure is followed for each training sample. Once the gradient is calculated, weights can be updated using stochastic gradient descent [nie, top]

1.3.3 Contrastive Divergence

The algorithm consists of basically three steps. In the positive phase step, input is fed to the network, which is propagated to the hidden layer. In the negative phase, the activation of the hidden layer is propagated back to the input layer. The produced output at the input layer is again propagated to the hidden layer, which results in another activation. In the third step, the weights are updated using the former and latter inputs and activations.

$$w(t+1) = w(t) + \gamma(\mathbf{v}\mathbf{h}^{\mathbf{T}} - \mathbf{v}'\mathbf{h}'^{\mathbf{T}}) \tag{1.7}$$

where v is the input, h is the first activation, v' is the produced input, accordingly h' is the second activation.
This algorithm aims to generate data close to the raw input. The idea behind the algorithm is to improve the reproduced data by adjusting weights and represent it as close as possible to the raw input [top].

1.4 Structures of Neural Networks

There are two main structures of neural networks: Probabilistic models, such as RBM and direct encoding models, such as autoencoder [mar].

1.4.1 Autoencoder

Autoencoders, which are feedforward neural networks, learn to produce a compressed, encoded data from the raw input. They tend to reconstruct the input using compression. The autoencoders do not aim to learn based on the training data and their labels, but rather aim to learn the internal structure of the input. In this manner, relatively small amount of neurons in the hidden layers learn the data conceptually to provide a compact representation. Therefore hidden layers in an autoencoder can be interpreted as feature detector. Autoencoders are powerful architectures for learning in the sense that they are helpful to overcome the issue of overfitting by providing compact representations of the raw input [top].

1.4.2 Restricted Boltzmann Machine (RBM)

RBM is a generative stochastic neural network, which learns a probability distribution over its set of inputs [Smo86].

In RBMs the connections between the layers are undirected, which means the values are propagated across the layers in both directions. RBMs differ from Boltzmann machines, since only sequential layers are connected to each other [mar].

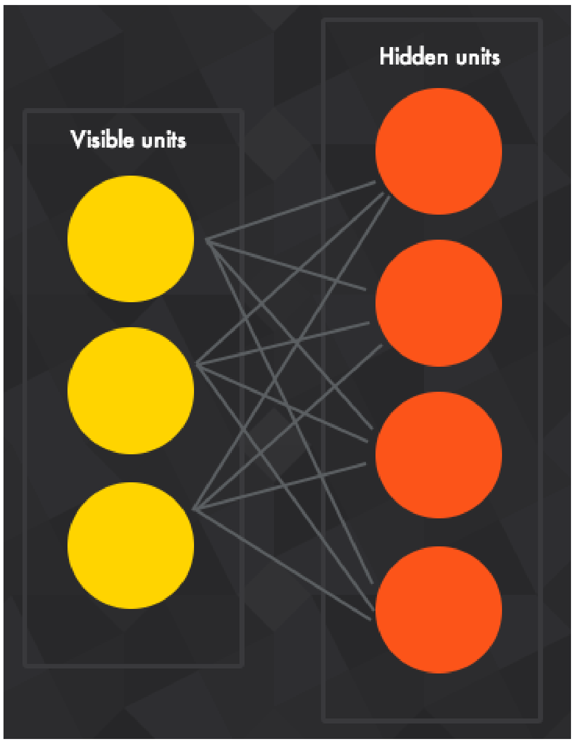


Figure 1.5: RBM model [top]

Contrastive divergence algorithm is used for the training of RBMs [top].

Chapter 2

Deep Learning

Deep learning is a popular research area especially in the neural networks and already provides applications in various domains such as computer vision, speech recognition, natural language processing, hand writing recognition etc. [SB13, Las11]. It has provided commercial success to technology companies like Google and Microsoft by providing speech recognition, image classification methods. Facebook is also planning to empower its big data with deep learning methods to make predictions about its users [Tec].

Deep learning approaches the data by representing it hierarchically [Den13]. The features are distributed in different layers and separated between the blocks. Because of the hierarchical method the level of the abstraction is increased while the input data is processed at each layer. A good example can be image recognition. In the lowest layer are pixels and at higher layers edges, patterns, parts and finally the recognized object [YL13].

It is possible to approximate any random function with a single hidden layer. Nevertheless it is easier to learn with multiple layers, which leads to deep networks. Deep networks consist of multiple hidden layers that learn abstract features of the input by creating internal representations. Layers of higher orders thus combine the patterns remarked by the lower layers and create even more internal representations independent of the raw input. However, having large number of hidden layers introduces issues, such as vanishing gradients and overfitting. For deep networks, backpropagation algorithm loses its power, since the gradients start vanishing and become relatively very small, as the error is backpropagated. In addition, deep networks can perform incompetent due to the overfitting, which is caused by letting the network learn the training samples too precise.

The autoencoders and RBMs can be stacked on top of each other to form deep neural networks. Using greedy layerwise pre-training algorithm can solve the problem of vanishing and overfitting problems [top].

2.1 Deep Neural Network (DNN) Architectures

2.1.1 Stacked autoencoders

The autoencoder idea states that not all features of the input are suitable for a specific task, such as classification. Encoder transforms the input vector into hidden representation and decoder maps the hidden representation back to reconstructed input. The autoencoding process compares the reconstructed input to the original input and tries to minimize the error to create the reconstructed value close to the original [PV10].

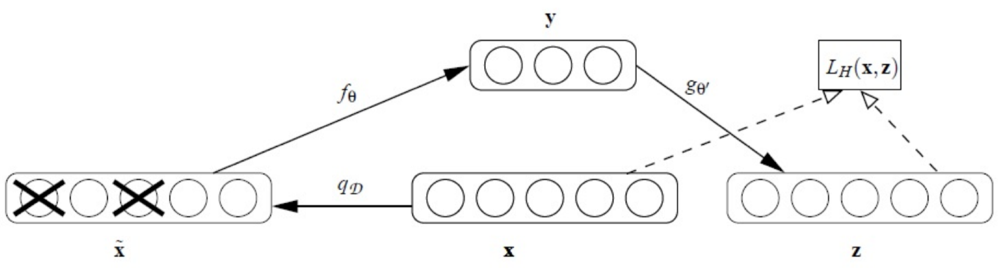


Figure 2.1: Denoising Autoencoder. Sample x stochastically corrupted via q_D to $\tilde{x}$. Then autoencoder maps it to $y$via encoder f_θ and reconstructs to z via decoder g_{theta}. Reconstruction error is measured by $L_H(x, z)$ [PV10]

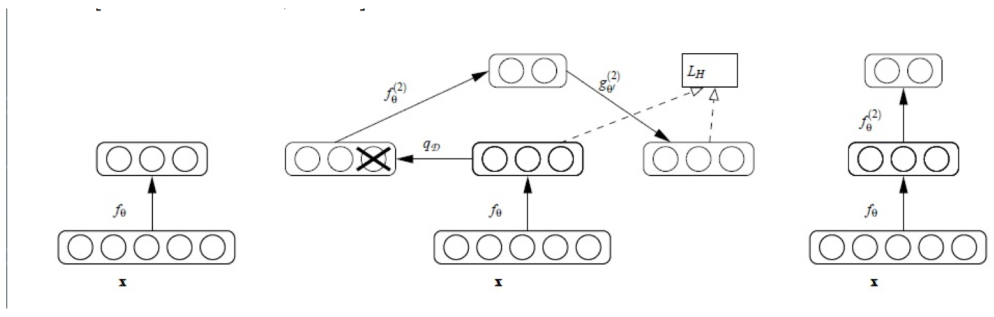


Figure 2.2: Stacked denoising autoencoder. After learning first encoding function, the raw input is first encoded via f_θ and corrupted via q_D. Afterwards same denoising autoencoder procedure as in Figure 2.1 to learn encoding function of the second layer. [PV10]

The greedy pre-training procedure for autoencoders is as follows:
An output layer is stacked on the first hidden layer of autoencoder. The input is given to the network, which propagates through the first hidden layer to the output layer. The network is trained using backpropagation with training samples. The output layer of the first hidden layer is then removed from the network.

An output layer is stacked on the second hidden layer of autoencoder. The input is given to the network, which propagates through the first and second hidden network to the output layer. The weights are again updated using backpropagation.

This procedure is repeated for all the layers. Once the weights are initialized, the network can be stacked an output layer and trained using backpropagation to produce desired output. This step is called fine-tuning [top].

In the case of denoising autoencoders, the useful features for a specific task are exalted and the influence of the rest is reduced (denoised). A good representation here would mean that the higher levels of representations are robust to the corruption of the input and extracting features for representation of the input is crucial [PV10].

The algorithm is as follows: A stochastically corrupted input is produced from the uncorrupted input. The encoder maps the corrupted input and then the decoder reconstructs the output of the encoder. A minimization algorithm obtains the reconstruction value close to the uncorrupted input. In short, the reconstruction function is a function of the corrupted input, which provides a close value to the uncorrupted input. For this architecture, the parameters are randomly initialized and adjusted using stochastic gradient descent algorithm [PV10].

Once the first encoder function is learned, the first encoder function is run on the uncorrupted clean input. The first resulting representation is used as the input of the auto-encoder in order to learn the encoder function of the second layer and this procedure is repeated for further layers [PV10].

2.1.2 Deep Belief Networks (DBN)

DBNs have one visible layer at the input and hidden layers up to the output. The blocks in the hidden layers learn the statistical representations via connections to the lower layers. The representations become more complex throughout the layers from bottom to the output [HL09].

Greedy layerwise unsupervised pre-training can be used for pre-training as learning algorithm in DBNs [PH10]. After pre-training, gradient descent method can be used for fine-tuning.

DBNs, as seen in Figure 2.3 [PH10], consist of restricted Boltzmann machines (RBMs) aligned on top of each other [GEH06]. Pre-training of DBN is done via greedy bottom-up only. This approach of greedy has a drawback; it assumes the lower-layers of the system remain unchanged while pre-training a layer [ZY13].

The greedy layerwise pre-training algorithm works as follows:

The first RBM is trained using contrastive divergence with training samples. The input samples are then given to input layer of first RBM, which propagate to the second RBM. The resulting output is used to start contrastive divergence training of the second RBM. After repeating this procedure for all RBMs, the network of RBMs can be stacked with the final RBM layer and fine-tuned using backpropagation [top].

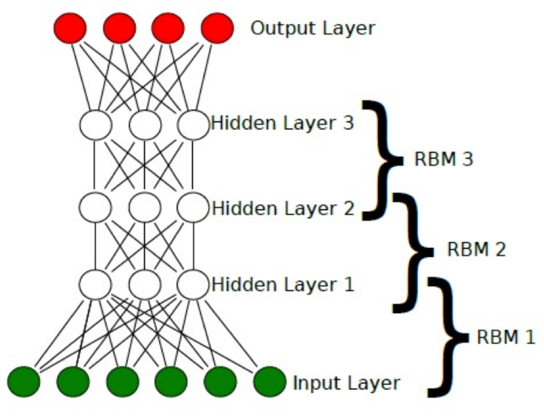


Figure 2.3: DBN model [PH10]

2.1.3 Deep Boltzmann Machine (DBM)

In DBMs top-down approach is also possible in addition to greedy bottom-up, which can provide transfer of the hidden information top-down. As can be seen in the Figure asd [RS10], there can be connections in both directions across the layers in DBM, whereas DBNs have directed connections bottom-up. Unlike DBNs, DBMs can adopt training bottom-up and top-down in order to provide better representations of complex input [RS09, YB07].

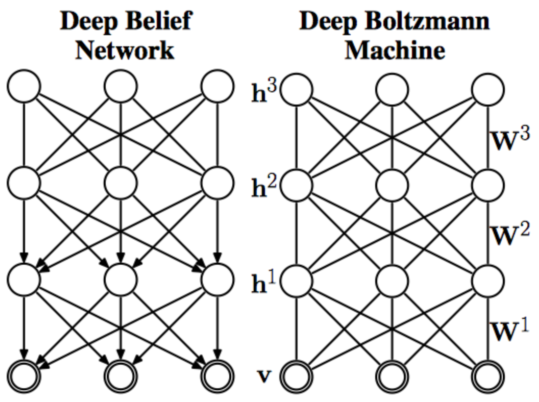


Figure 2.4: Three layer DBN (left) and DBM (right), W denote the weights and h the activations [RS09]

In practice the pre-training algorithm works well. Nevertheless, it requires bringing up a model with symmetric weights only and does not provide information about what has happened during pre-training [HS12].
Despite their advantages, DBMs are slower than single bottom-up DBNs, which complicates the usage for large data sets [RS10].

2.1.4 Convolutional Neural Networks

Convolutional neural networks are designed to work directly on the input with minimal pre-processing [ARC09].
Multilayer Perceptrons, MLPs, can show great performance for simple data structures but also lack in some situations, such as distortion and high amounts of training parameters. Having a large number of connections per single neuron in the hidden layer will be hard to train. To overcome these issues, classification has been considered in two processes: feature extraction and classification itself. Feature extraction is a relatively hard process, for which experts should provide suitable features to extract [LBBH98].
Convolutional neural networks are variations of MLP architectures and are inspired by biological processes [HW62]. They consist of multiple layers of neuron groups, which are related to portions of the input data. The results from these groups are then aligned to overlap in order to tolerate translation of the input data [KMN+03].
The advantage of convolutional networks is that the layers share the weights of connections, which reduces memory requirement and improves performance [Lec].
Convolutional neural networks are mostly used in image recognition, where the learning process can be very fast [CMM+11]. There are also applications for facial recognition, where the error rate has been considerably reduced [LGTB97].

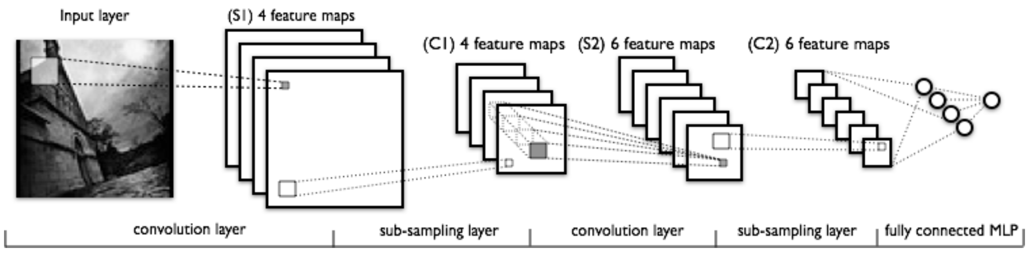


Figure 2.5: LeNet, convolutional network for image recognition [dee]

Convolutional networks are mostly used in image recognition. The procedure of image recognition with convolutional networks is complex, but to give an idea, it can be summed up under these steps [top]:

- The input image is convolved using a filter. The predefined filter for a specific task is a pixel window with weights. It crawls through the input image shifting its window. While convolving, pixels under the input window and the filter matrix are multiplied to create the feature map (FM).
- A convolutional layer can contain many filters. Thus, the weights of the connections are shared among the neurons, which belong to different filters in the layer.
- On top of the convolutional layers are the subsampling layers, which take the FMs as input and reduce their resolution by subsampling them. There are

several subsampling methods. The most popular are average pooling, max pooling and stochastic pooling.

- The network is usually stacked with additional higher layers to introduce the targeted representation from the convolution process. The training of the convolutional neural networks is done via backpropagation algorithm, by updating the filter weights.

2.1.5 Deep Stacking Networks (DSN)

DSNs are attractive due to their advantage, which is parallel learning of the weights. Simple blocks of classifiers are produced first and then stacked on top of each other to learn more complex classifiers [Wol92].

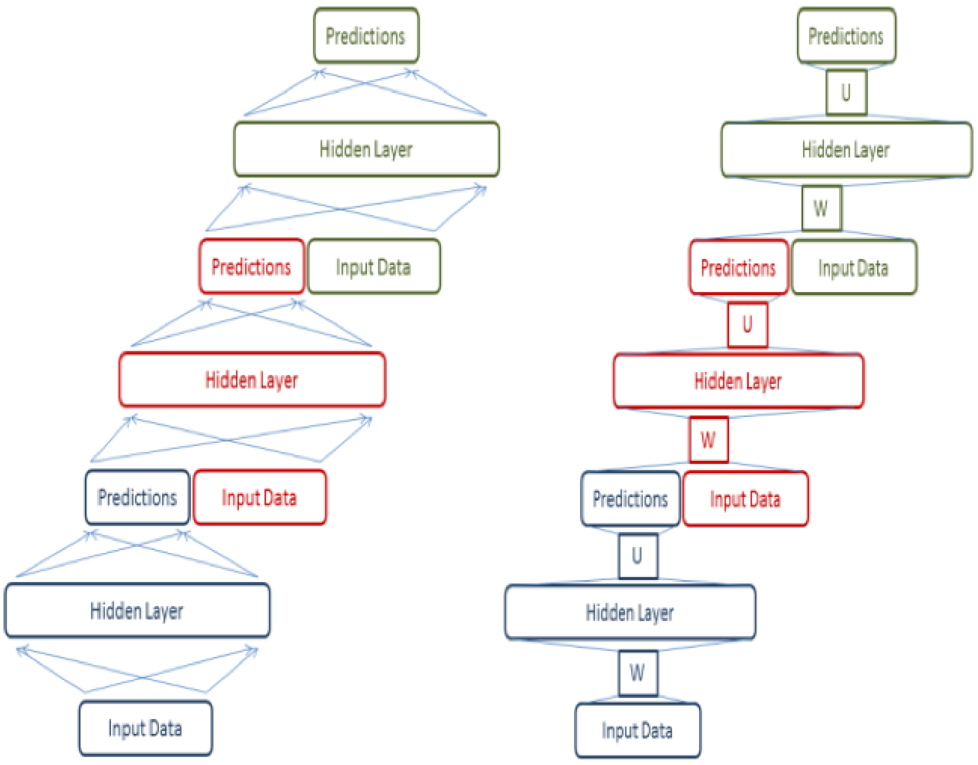


Figure 2.6: DSN mechanism model [DHY12]

2.1.6 Compound Hierarchical-Deep Models

The architectures described above can supply good representations for fast and proper classification tasks with high dimensional training data. In these architectures the neurons deal with representing the input throughout the network. Nevertheless, these architectures are not powerful in case of learning with few training data. Hierarchical Bayesian (HB) model makes it possible to learn with few training data [FFFP06]. However HB models do not produce representations with unsupervised methods [STT13]. Compound HD architectures provide both of HB and deep network characteristics.

2.1.7 Convolutional Deep Belief Networks

Convolutional DBNs are particularly useful for high dimensional data and are powerful for image processing [LPLN09]. They can scale elegantly to realistic images and are also translation invariant [LGRN09]. Convolutional DBNs are similar to DBN and also trained with greedy bottom-up method. In addition to DBNs, the weights between visible and hidden layers are shared among the blocks in the hidden layer.

2.1.8 Deep Long-Short Term Memory Neural Networks (LSTM)

In this architecture, LSTM replaces RBM in a recurrent network, which have the ability of memorizing. LSTM block has input, output and forget gates, that are responsible for write, read and reset operations on the memory. The block can memorize for long time and thus overcome vanishing gradient issue [Gra12b]. LSTM architectures are especially useful for speech recognition [GFGS06]. An LSTM block has 4 input terminals and only 1 output terminal.
Figure [Gra12b].
Comparison Figure graves2013speech.
For problems involving sequential learning, such as phoneme classification, LSTM networks are approved to perform better than other architectures [GMH13].

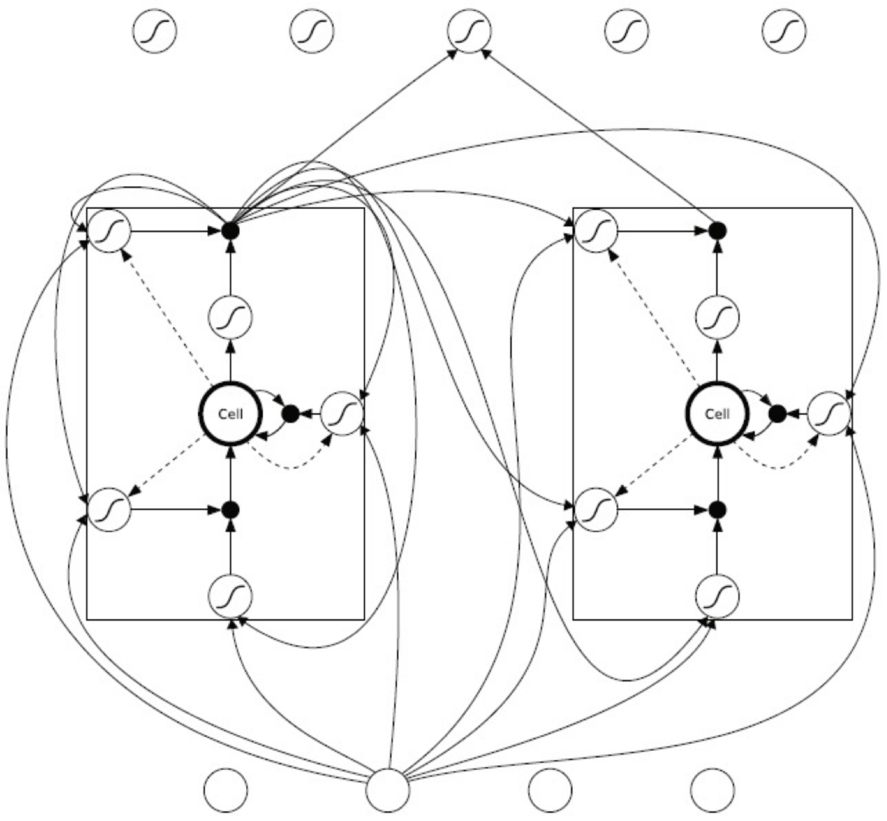


Figure 2.7: An LSTM network, which consists of four inputs, a hidden layer of two LSTM memory units and five outputs. [Gra12b]

Method	PER
Sequential DBN	24.2%
DBN-HMM (Hybrid Model)	23.0%
DBN with mcRBM feature extraction (Hybrid Model)	20.5%
Deep Long-Short Term Memory RNNs with CTC	18.4%
Deep Long-Short Term Memory RNNs with transducer	**17.7%**

Figure 2.8: Benchmarks of phoneme recognition on TIMIT data set with different methods. PER denotes phoneme error rate [GMH13]

Chapter 3

Spiking Neural Networks (SNN)

In traditional neural networks, a neuron generates a spike, the action potential, at the kernel of the cell, soma. This spike in the form of a 1-2ms pulse travels through the axon, which is linked to the dendrite of another neuron via synapses. The incoming spike influences the receiving neuron's membrane potential, which causes the neuron to fire a spike. A spike can have either a positive or a negative impact on the receiving neuron, also called postsynaptic neuron. The positive one is called postsynaptic potential (PSP) and the negative one is inhibitory postsynaptic potential (IPSP) [PMB12].

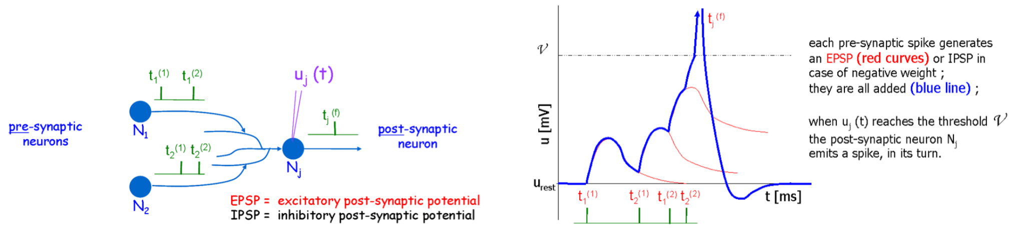

pre-synaptic neurons
N1
N2
t1(1) t1(2)
t2(1) t2(2)
uj (t)
tj(f)
Nj
post-synaptic neuron
EPSP = excitatory post-synaptic potential
IPSP = inhibitory post-synaptic potential
𝒱
u [mV]
urest
tj(f)
t1(1) t2(1) t1(2) t2(2)
t [ms]
each pre-synaptic spike generates an EPSP (red curves) or IPSP in case of negative weight ; they are all added (blue line) ;
when uj (t) reaches the threshold 𝒱 the post-synaptic neuron Nj emits a spike, in its turn.

Figure 3.1: Spiking neuron model. N_j fires when the weighted sum of incoming stimulations is above the threshold. Membrane potential is seen on the right [PMB12]

According to the discoveries in the field of neuroscience, the timing and number of the spikes code the information carried by neurons [TFM$^+$96, VRT01]. Humans can respond to a visual stimulation in just 100-150ms. Assuming that a feedforward neural network with average firing rate of 10ms provides the visual processing, it would be hardly possible to send single spike. Thus it is more likely that the information is encoded via timing also [O$^+$96].

In SNNs the neurons rely on the timing of the spikes in order to communicate with each other. Since the basic principle of SNNs is different than traditional neural networks, it is required to adapt the learning rules. For doing that, it is crucial to define a model for a neuron.

3.1 Models of spiking neurons

A model of a neuron provides a formulation how a neuron processes the incoming spikes and triggers firing accordingly. There are many models to describe these models and common ones are explained.

Hodgkin-Huxley (HH) model is based on the conductance of ions channels [HH52]. It models the neuron using capacitors and conductance values of ion channels and their equilibrium potentials.

$$C\frac{du}{dt} = -g_{Na}m^3h(u-E_{Na}) - g_Kn^4(u-E_K) - g_L(u-E_L) + I(t) \tag{3.1}$$

$$\tau_n\frac{dn}{dt} = [n-n_0(u)], \tau_m\frac{dm}{dt} = -[m-m_0(u)], \tau_h\frac{dh}{dt} = -[h-h_0(u)] \tag{3.2}$$

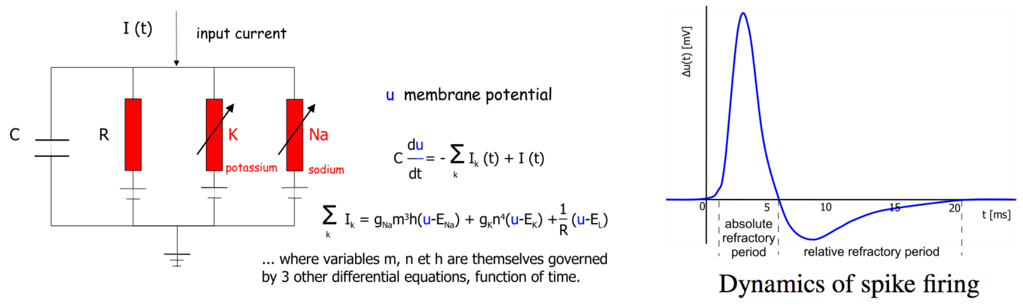

I (t)
input current
u membrane potential
C
R
K
potassium
Na
sodium
$C\frac{du}{dt} = -\sum_k I_k(t) + I(t)$
$\sum_k I_k = g_{Na}m^3h(u-E_{Na}) + g_Kn^4(u-E_K) + \frac{1}{R}(u-E_L)$
... where variables m, n et h are themselves governed by 3 other differential equations, function of time.
Δu(t) [mV]
0
5
10
15
20
t [ms]
absolute refractory period
relative refractory period
Dynamics of spike firing

Figure 3.2: HH model. Electrical circuit (left) and membrane potential (right. According to the model, also the voltage across the capacitor) [PMB12]

By these equations it is possible to model very detailed, such as sudden increase of potential at firing, absolute refractoriness period and relative refractory period. Since HH is too complex, it is hard to realize large networks even for an SNN simulation.

Integrate-and-Fire (IF) is derived from HH and has less computational costs. IF has variants, such as Leaky-Integrate-and-Fire (LIF) [Orh12]. LIF ignores the shape of the action potential and pays attention to the timing of the spike.

$$\tau_m\frac{du}{dt} = u_{rest} - u(t) + RI(t) \tag{3.3}$$

The action potential is reset to u_{rest} immediately after firing. The absolute refractory period can be defined as $u = -u_{abs}$ for a period after firing and then setting the action potential to u_{rest}.
Quadratic-Integrate-and-Fire (QIF) model depends on the square of the action potential. In addition to LIF, QIF model is able to provide dynamic spiking properties, such as delayed spiking and activity dependent thresholding.

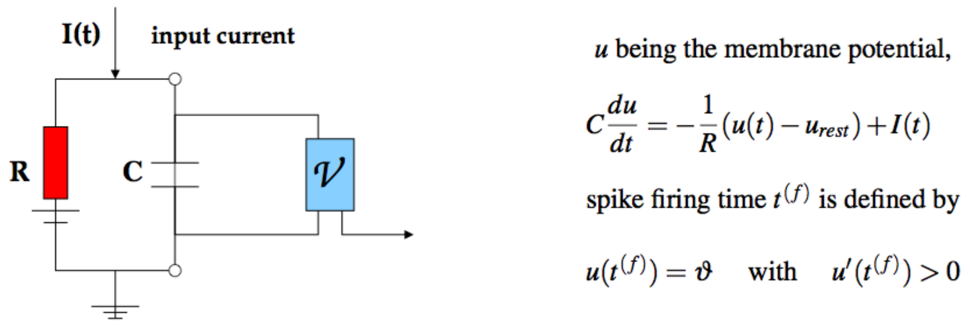

I(t)
input current
R
C
$\mathcal{V}$
u being the membrane potential,
$C\frac{du}{dt} = -\frac{1}{R}(u(t) - u_{rest}) + I(t)$
spike firing time $t^{(f)}$ is defined by
$u(t^{(f)}) = \vartheta$ with $u'(t^{(f)}) > 0$

Figure 3.3: IF neuron model [PMB12]

Theta neuron model can be interpreted as a transformation of QIF model, where the state is determined by a phase. As the neuron fires a spike, the phase crosses pi [PMB12].

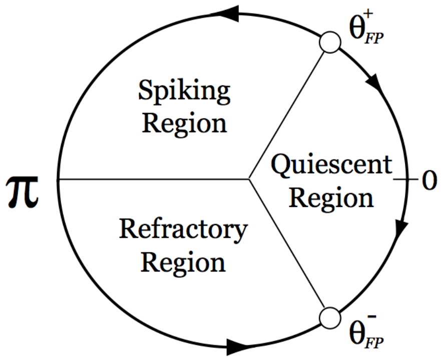


Figure 3.4: Theta model, the phase circle. Phase crosses π when the neuron fires. Input above threshol moves the phase from the attractor point to the saddle point [PMB12]

$$\frac{d\theta}{dt} = (1 - cos(\theta)) + \alpha I(t))(1 + cos(\theta)), \tag{3.4}$$

where θ is the neuron phase, α the scaling constant and $I(t)$ the input current.

Izhikevich model provides a balance between computational cost and biological reality. It is indeed possible to describe many different firings using Izhikevich model [I$^+$03].

Spike Response Model (SRM) defines the action potential as an integral over the past with consideration of the refractoriness period. It is based on the occurrence of firings [Ger95, KGH97]. SRM is able to simulate complex calculations, even though it has a remarkably simpler structure than HH model [PMB12].

3.2 Synaptic Plasticity

Synaptic plasticity means the weight adjustment of the synapses, which is the basis of learning. The case of weight strengthening is called potentiation and weakening depression. If the impact of a change lasts up to a few hours, it is called long term and up to a few minutes scale is called short term. To summarize there are for possibilities of an impact of a weight change: Long Term Potentiation (LTP) or Depression (LTP), Short Term Potentiation (STP) or Depression (STD). Synaptic plasticity depends on the presence and precise timing of the spikes [MLFS97, BP98, KGVH99].

3.2.1 Spike-Timing Dependent Plasticity (STDP)

According to STDP, the weight is strengthened, if postsynaptic firing occurs after presynaptic firing. As appropriate, the weight is decreased if presynaptic firing occurs after postsynaptic firing. In case the post- and presynaptic firings are distant to each other in time, the weights are not changed. The change of weight in the STDP can be modeled as a function of the difference between post- and presynaptic firing [PMB12].

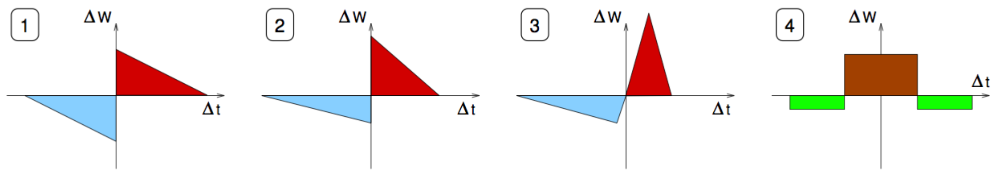


Figure 3.5: Several STDP shapes that describe the adjustment of the synaptic weight according to the delay. [PMB12]

3.3 Computational power and complexity of SNN

SNN has two major advantages over traditional networks: Fast real-time decoding of signals and high information carriage capacity via adding temporal dimension [PMB12, TFM$^+$96].
The spikes of the 3rd generation neural networks are categorized into two types, type A and B. The type A neuron's spike shows the behavior of a binary activation function with delay and has a rectangular shape. The type B has a triangular shape with delay variable, as well. In order to realize networks with real valued inputs the type B is considered due to its shape. Concerning this, a type B neuron can shift the firing time of a postsynaptic neuron [Maa97b].
An SNN can realize computations like traditional neural networks of similar architectures without taking temporal coding into account. SNN gains power from its additional dimension, the delay variable [Maa97a].

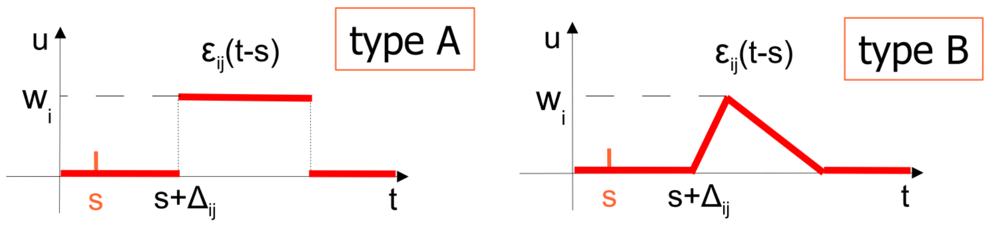


Figure 3.6: TypeA and Type B activation functions. s is the time of the received spike, w is the synaptic weight. The neuron fires with an axonal delay. [PMB12]

Using SNN with noisy neurons, feedforward neural network of second generation can be simulated in temporal coding and with high reliability [Maa01]. Thus SNN are defined as universal approximators [PMB12].
STDP learning lets a spiking neuron to realize any mapping of spike trains [LNM05]. In addition, the amount of samples required for SNN is not remarkably larger than traditional neural networks [Sch04]. However, learning of a spiking neuron is more complex than neurons of the other generations [Maa01]. Therefore, it is crucial to develop mechanisms to adapt complex biological neural systems, especially the computations regarding continuously changing time [PMB12].

3.4 Structures with SNN

3.4.1 Cell assembly and Polychronization

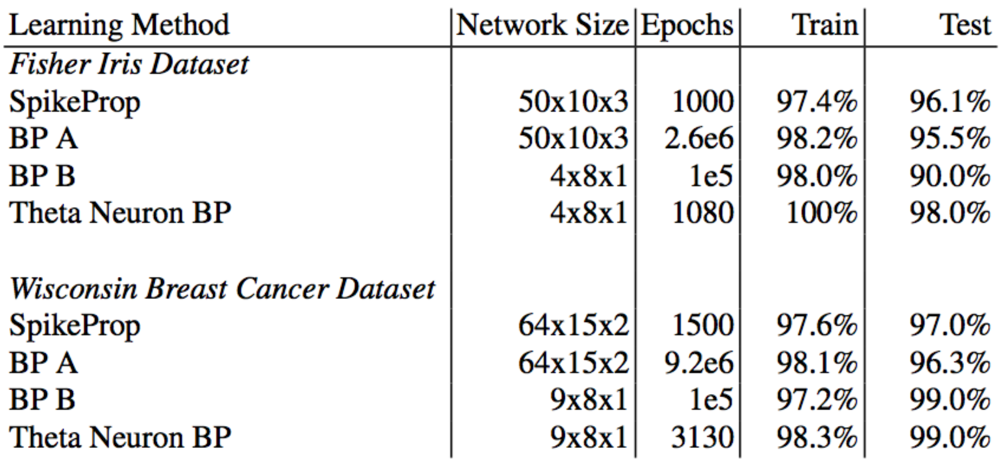
New technologies in brain imaging made it possible to observe several neurons simultaneously. A cell assembly is a group of neurons with mutual activities, which can also be interpreted as an operational unit. Cell assemblies are the basis neural blocks of the memory, such that resonant activities cause short-term and the formation of the cell assemblies produces long-term memory [Heb02, PMB12].
A polychronous group of neurons are connected to each other with varying delays. Stimulation of the members via a specific spike pattern leads to the activation of the polychronous group. A neuron can belong to several polychronous groups, which provide a great information capacity due to the existence of varying delays [Izh06]. Conceptually, polychronization is indeed a computational implementation of cell assemblies [PMB12].

3.5 Learning with SNN

The supervised and unsupervised learning methods for traditional neural networks can be adapted to SNN, where algorithms use temporal coding and adjust the weights according to the delays. In addition, it is possible to develop learning algorithms purely for SNN, which consider temporal domain and the complexity of SNNs for temporal pattern detection [PMB12].

3.5.1 Traditional learning models with SNN

Two layers of SRM neurons with multiple synapses between neuron pairs, accordingly various delays and weights, build an SNN to perform unsupervised clustering by taking spike-times as input [BLPK02]. In compliance with time-variant of Hebbian learning rule, the weight of a synapse is increased significantly, if the firing of the target neuron follows it. Accordingly, earlier and later synapses' weights are decreased [NR98].
SpikeProp, a supervised learning algorithm for SNNs and derived from backpropagation, uses SRM neurons to transfer the information in spike-times. The neuron pairs in this algorithm also have multiple delayed synapses with weights [MVB09]. Extensions on the SpikeProp provide adaption of delays across the gradient error and faster convergence of the algorithm [SVC04, XE01].
Theta Neuron learning, another supervised learning method creates a continuous and cyclic model via mapping QIF neurons to Theta neuron [MVB09].



Learning Method	Network Size	Epochs	Train	Test
Fisher Iris Dataset				
SpikeProp	50x10x3	1000	97.4%	96.1%
BP A	50x10x3	2.6e6	98.2%	95.5%
BP B	4x8x1	1e5	98.0%	90.0%
Theta Neuron BP	4x8x1	1080	100%	98.0%
Wisconsin Breast Cancer Dataset				
SpikeProp	64x15x2	1500	97.6%	97.0%
BP A	64x15x2	9.2e6	98.1%	96.3%
BP B	9x8x1	1e5	97.2%	99.0%
Theta Neuron BP	9x8x1	3130	98.3%	99.0%

Figure 3.7: Comparison table of classifications by several learning methods on two different data sets. BPA and B denote standard Matlab implementations [MVB09]

Both of these gradient-based methods require fine-tuning, especially to promote survival of the neurons, which stop firing spikes. By adjusting the learning rates, this issue has been solved [MVB09].

3.5.2 Reservoir Computing

Biologically inspired structures, such as networks with sparsely spread neurons and varying spikes; can be adapted to SNN architectures. Reservoir Computing defines a family of networks, which process temporal patterns with 3rd generation neurons. The architecture of a Reservoir Computing network consists of three main parts: the input layer to the reservoir, the reservoir itself, which is a recurrent network, and the output layer.

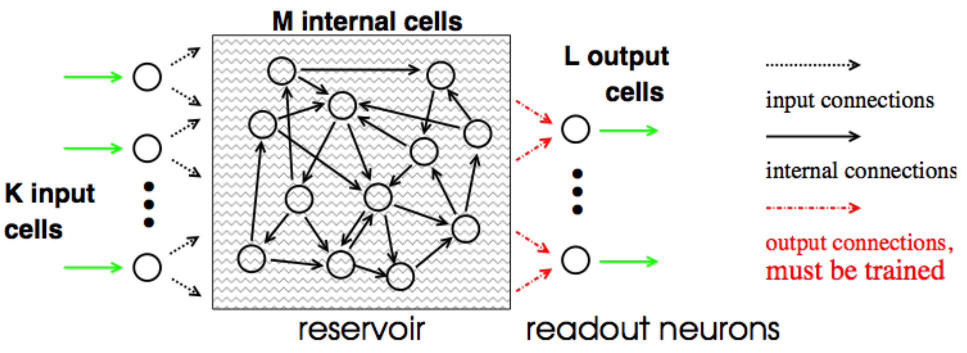


Figure 3.8: Reservoir Network sample [PMB12]

The main idea of reservoir computing is to select the relevant part from the reservoir with neurons of the output layer. STDP is used to train the reservoir and the connections from the reservoir to output layer are trained via linear regression [PMB12].
Echo State Network (ESN) and Liquid State Machine (LSM) are both models of reservoir computing, which are convenient to exploit temporal features of spiking neurons.

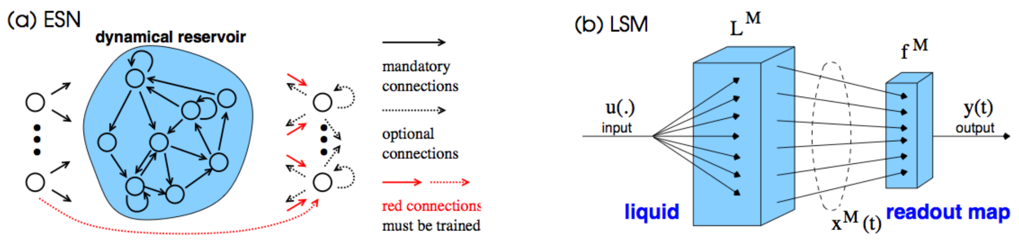


Figure 3.9: ESN (a) and LSM (b), $x^M(t)$ denotes ”liquid state” [PMB12]

ESN aims to learn time series using recurrent networks. The internal state of the reservoir is a function of the concurrent input, former state and former output. An efficient ESN should be able to forget, which is possible via adjusting the reservoir weights, such that the behavior of fading memory is realized. Fading memory can be implemented by choosing weight matrix with absolute eigenvalues lower than 1 [Jae01].
LSM concentrate on processing continuous input in in real-time using IF neurons. The reservoir, also called ”liquid filter” transforms the input to ”liquid state”, from

which the output is generated using a "memoryless readout map". The readout map is supposed to produce stable output from the reservoir [101]. Having several readout maps may enable parallel real-time computing [PMB12].
Reservoir computing architectures are convenient for time series prediction and temporal pattern recognition. Both of LSM and ESN may process temporally changing information [PMB12].
Usage of LSM usually takes place in biologically inspired models. Since SNN has temporal dynamics, it is suitable for sequential data. LSMs with spiking neurons may thus lead to satisfying models in speech recognition [VSS05]. Accordingly, SNN architectures may also provide solutions in computer vision using spike asynchrony [TG97].

3.6 Applications of SNN

The research around SNN is not limited to reservoir computing. There are attempts to make use of SNN for different purposes, for instance to provide methods like PCA [BM08]. These attempts promoted progress, for example in efficient encoding of auditory, which outperforms standard filters [SL06]. Due to their fast processing ability, SNN systems can be used in the field of autonomous robotics as well [MSK02, PMB12].
SNN systems benefit from parallel computing, since SNNs have a better balance between communication time and computation cost than traditional networks. In SNN systems only the weight values of active neurons need to be accessed for further calculations and computing the membrane potential for a spiking neuron is a relatively complex operation as compared to weighted sum [PMB12]. SpikeNET approves the performance of parallel computing by SNN systems [DGvRT99].
There are several simulators of spiking neurons, basically categorized as time-driven and event-driven simulators. Time-driven simulators, such as GENESIS and NEURON [HC97], are used for biophysical models, whereas event-driven simulators are better suitable for fast simulation of large SNNs, such as MVASpike [MVA], event-driven NEURON [HC04] and DAMNED [MPMP$^+$06].
The python package pyNN stands out for development purposes, where the neural network models can easily be simulated on several simulators. Inspired by the human brain, there are attempts to implement hardware SNNs. Neurogrid and SpiNNaker are popular hardware simulators of SNN. Neurogrid is able to combine parallel operation and programmability under low power consumption [Neu]. SpiNNaker, which is built from ARM processors, aims to provide a massively parallel computing platform based on a six-layer thalamocortical model [FGTP14].

Bibliography

[And95] J. A. Anderson. *An introduction to neural networks.* MIT Press, 1995.

[ARC09] Itamar Arel, Derek Rose, and Robert Coop. Destin: A scalable deep learning architecture with application to high-dimensional robust pattern recognition. In *AAAI Fall Symposium: Biologically Inspired Cognitive Architectures*, 2009.

[BLPK02] Sander M Bohte, Han La Poutré, and Joost N Kok. Unsupervised clustering with spiking neurons by sparse temporal coding and multilayer rbf networks. *Neural Networks, IEEE Transactions on*, 13(2):426–435, 2002.

[BM08] Lars Buesing and Wolfgang Maass. Simplified rules and theoretical analysis for information bottleneck optimization and pca with spiking neurons. In *Advances in Neural Information Processing Systems*, pages 193–200, 2008.

[BP98] Guo-qiang Bi and Mu-ming Poo. Synaptic modifications in cultured hippocampal neurons: dependence on spike timing, synaptic strength, and postsynaptic cell type. *The Journal of neuroscience*, 18(24):10464–10472, 1998.

[CMM$^+$11] Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

[dee] URL: `http://deeplearning.net/tutorial`.

[Den13] L. Deng. Three classes of deep learning architectures and their applications: A tutorial survey. 2013. URL: `research.microsoft.com`.

[DGvRT99] Arnaud Delorme, Jacques Gautrais, Rufin van Rullen, and Simon Thorpe. Spikenet: A simulator for modeling large networks of integrate and fire neurons. *Neurocomputing*, 26:989–996, 1999.

[DHY12] Li Deng, Brian Hutchinson, and Dong Yu. Parallel training for deep stacking networks. In *INTERSPEECH*, 2012.

[FFFP06] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 28(4):594–611, 2006.

[FGTP14] Steve B Furber, Francesco Galluppi, Steve Temple, and Luis A Plana. The spinnaker project. 2014.

[GEH06] Y.W. Teh G. E. Hinton, S. Osindero. A fast learning algorithm for deep belief nets. neural computation. *Neural Computation*, 18(7):1527–1554, 2006.

[Ger95] Wulfram Gerstner. Time structure of the activity in neural network models. *Physical review E*, 51(1):738, 1995.

[GFGS06] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[GMH13] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6645–6649. IEEE, 2013.

[Gra12a] A. Graves. *Supervised sequence labelling with recurrent neural networks.* Springer, 2012.

[Gra12b] Alex Graves. *Supervised sequence labelling with recurrent neural networks*, volume 385. Springer, 2012.

[HC97] Michael L Hines and Nicholas T Carnevale. The neuron simulation environment. *Neural computation*, 9(6):1179–1209, 1997.

[HC04] ML Hines and Nicholas T Carnevale. Discrete event simulation in the neuron environment. *Neurocomputing*, 58:1117–1122, 2004.

[Heb02] Donald Olding Hebb. *The organization of behavior: A neuropsychological theory.* Psychology Press, 2002.

[HH52] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500, 1952.

[HL09] A. Y. Ng H. Lee, P. Pham. *Unsupervised feature learning for audio classification using convolutional deep belief networks, Advances in Neural Information Processing Systems 22.* 2009.

[HS12] Geoffrey E Hinton and Ruslan Salakhutdinov. A better way to pretrain deep boltzmann machines. In *Advances in Neural Information Processing Systems*, pages 2447–2455, 2012.

[HW62] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[I$^+$03] Eugene M Izhikevich et al. Simple model of spiking neurons. *IEEE Transactions on neural networks*, 14(6):1569–1572, 2003.

[Izh06] Eugene M Izhikevich. Polychronization: computation with spikes. *Neural computation*, 18(2):245–282, 2006.

[Jae01] Herbert Jaeger. The "echo state" approach to analysing and training recurrent neural networks-with an erratum note. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148:34, 2001.

[Kal09] J. W. Kalat. *Biological psychology.* Wadsworth, Cengage Learning, 2009.

[KGH97] Werner Kistler, Wulfram Gerstner, and J Hemmen. Reduction of the hodgkin-huxley equations to a single-variable threshold model. *Neural Computation*, 9(5):1015–1045, 1997.

[KGVH99] Richard Kempter, Wulfram Gerstner, and J Leo Van Hemmen. Hebbian learning and spiking neurons. *Physical Review E*, 59(4):4498, 1999.

[KMN$^+$03] Keisuke Korekado, Takashi Morie, Osamu Nomura, Hiroshi Ando, Teppei Nakano, Masakazu Matsugu, and Atsushi Iwata. A convolutional neural network vlsi for image recognition using merged/mixed analog-digital architecture. In *Knowledge-Based Intelligent Information and Engineering Systems*, pages 169–176. Springer, 2003.

[Las11] J. Laserson. From neural networks to deep learning. *XRDS: Crossroads, The ACM Magazine for Students*, 1(18):29, 2011.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[Lec] Y. Lecun. Lenet-5, convolutional neural networks. URL: `http://yann.lecun.com/exdb/lenet/`.

[LGRN09] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 609–616. ACM, 2009.

[LGTB97] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back. Face recognition: A convolutional neural-network approach. *Neural Networks, IEEE Transactions on*, 8(1):98–113, 1997.

[LNM05] Robert Legenstein, Christian Naeger, and Wolfgang Maass. What can a neuron learn with spike-timing-dependent plasticity? *Neural computation*, 17(11):2337–2382, 2005.

[LPLN09] Honglak Lee, Peter Pham, Yan Largman, and Andrew Y Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in neural information processing systems*, pages 1096–1104, 2009.

[Maa97a] Wolfgang Maass. Fast sigmoidal networks via spiking neurons. *Neural Computation*, 9(2):279–304, 1997.

[Maa97b] Wolfgang Maass. Networks of spiking neurons: the third generation of neural network models. *Neural networks*, 10(9):1659–1671, 1997.

[Maa01] Wolfgang Maass. On the relevance of time in neural computation and learning. *Theoretical Computer Science*, 261(1):157–178, 2001.

[mar] URL: `http://markus.com/deep-learning-101`.

[MLFS97] Henry Markram, Joachim Lübke, Michael Frotscher, and Bert Sakmann. Regulation of synaptic efficacy by coincidence of postsynaptic aps and epsps. *Science*, 275(5297):213–215, 1997.

[MPMP$^+$06] Anthony Mouraud, Hélène Paugam-Moisy, Didier Puzenat, et al. A distributed and multithreaded neural event driven simulation framework. In *Parallel and Distributed Computing and Networks*, pages 212–217, 2006.

[MSK02] Wolfgang Maass, Gerald Steinbauer, and Roland Koholka. Autonomous fast learning in a mobile robot. In *Sensor based intelligent robots*, pages 345–356. Springer, 2002.

[MVA] URL: `http://mvaspike.gforge.inria.fr`.

[MVB09] Sam McKennoch, Thomas Voegtlin, and Linda Bushnell. Spike-timing error backpropagation in theta neuron networks. *Neural computation*, 21(1):9–45, 2009.

[Neu] URL: `http://web.stanford.edu/group/brainsinsilicon/neurogrid.html`.

[nie] URL: `http://neuralnetworksanddeeplearning.com`.

[NR98] Thomas Natschläger and Berthold Ruf. Spatial and temporal pattern analysis via spiking neurons. *Network: Computation in Neural Systems*, 9(3):319–332, 1998.

[O^+96] Bruno A Olshausen et al. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996.

[Orh12] Emin Orhan. The leaky integrate-and-fire neuron model, 2012.

[PH10] D. Eck P. Hamel. Learning features from music audio with deep belief networks. *ISMIR*, 2010.

[PMB12] Hélene Paugam-Moisy and Sander Bohte. Computing with spiking neuron networks. In *Handbook of natural computing*, pages 335–376. Springer, 2012.

[PV10] I. Lajoie P. Vincent, H. Larochelle. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 11:3371–3408, 2010.

[RCO00] Y. Munakata R. C. O'Reilly. *Computational Explorations in Cognitive Neuroscience*. MIT Press, 2000.

[RS09] G. Hinton R. Salakhutdinov. Deep boltzmann machines. *International Conference on Artificial Intelligence and Statistics*, pages 448–455, 2009.

[RS10] H. Larochelle R. Salakhutdinov. Efficient learning of deep boltzmann machines. *International Conference on Artificial Intelligence and Statistics*, 2010.

[Sam07] S. Samarasinghe. *Neural networks for applied sciences and engineering: from fundamentals to complex pattern recognition*. Auerbach Publications, 2007.

[SB13] H. Larochelle H. Lee R. Salakhutdinov S. Bengio, L. Deng. Special section on learning deep architectures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(35):1795–1797, 2013.

[Sch04] Michael Schmitt. On the sample complexity of learning for networks of spiking neurons with nonlinear synaptic interactions. *Neural Networks, IEEE Transactions on*, 15(5):995–1001, 2004.

[SL06] Evan C Smith and Michael S Lewicki. Efficient auditory coding. *Nature*, 439(7079):978–982, 2006.

[Smo86] P. Smolensky. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition.* MIT Press, 1986.

[STT13] Ruslan Salakhutdinov, Joshua B Tenenbaum, and Antonio Torralba. Learning with hierarchical-deep models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(8):1958–1971, 2013.

[SVC04] Benjamin Schrauwen and Jan Van Campenhout. Improving spikeprop: enhancements to an error-backpropagation rule for spiking neural networks. In *Proceedings of the 15th ProRISC workshop*, volume 11, 2004.

[Tec] URL: `http://www.technologyreview.com/news/519411/facebook-launches-advanced-ai-effort-to-find-meaning-in-your-posts/`.

[TFM$^+$96] Simon Thorpe, Denis Fize, Catherine Marlot, et al. Speed of processing in the human visual system. *nature*, 381(6582):520–522, 1996.

[TG97] Simon J Thorpe and Jacques Gautrais. Rapid visual processing using spike asynchrony. *Advances in neural information processing systems*, pages 901–907, 1997.

[top] URL: `http://www.toptal.com/machine-learning/an-introduction-to-deep-learning-from-perceptrons-to-deep-networks`.

[VRT01] Rufin Van Rullen and Simon J Thorpe. Rate coding versus temporal order coding: what the retinal ganglion cells tell the visual cortex. *Neural computation*, 13(6):1255–1283, 2001.

[VSS05] David Verstraeten, Benjamin Schrauwen, and Dirk Stroobandt. Isolated word recognition using a liquid state machine. In *ESANN*, volume 5, pages 435–440, 2005.

[Wol92] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.

[XE01] Jianguo Xin and Mark J Embrechts. Supervised learning with spiking neural networks. In *Neural Networks, 2001. Proceedings. IJCNN'01. International Joint Conference on*, volume 3, pages 1772–1777. IEEE, 2001.

[YB07] Y. Lecun Y. Bengio. *Scaling learning algorithms towards AI, Large-Scale Kernel Machines.* 2007.

[YL13] M. Ranzato Y. Lecun. Deep learning tutorial, 2013. URL: `http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf`.

[ZY13] B. Xu Z. You, X. Wang. Investigation of deep boltzmann machines for phone recognition. *Acoustics, Speech and Signal Processing (ICASSP)*, pages 7600–7603, 2013.