Chapter 2 Introduction to Neural Networks

Abstract In this chapter we discuss how the biological neurons process information, the difference between the spatiotemporal processing of frequency encoded information conducted by a biological neuron and the amplitude and frequency encoded signals processed by the artificial neural networks. We discuss the various types of artificial neural networks that exist, their architectures and topologies, and how to allow such neural networks to possess plasticity, which allows the neurons to adapt and change as they process presynaptic signals.

Our brains are biological parallel computers, composed of roughly 100,000,000,000 (one hundred billion) signal processing elements called Neurons. Like a vast graph, these neurons are connected with each other in complex topological patterns. Each neuron in this vast processing graph accepts signals from thousands of other neurons, processes those signals, and then outputs a frequency encoded signal and passes it onwards to thousands of other neurons. Though each neuron on its own is relatively easy to understand, when you connect together a few billion of them, it becomes incredibly difficult to predict the outcome given some specific input. If you are careful and connect these biological signal processing elements in some particular pattern, the final output of this vast graph might even be something useful, an intelligent system for example. An output signal can for example control muscle tissue in your legs, so that they move in synchrony and give you the ability to walk and run. Or this vast neural network's output can be a solution to some problem which was fed into it as an image from its sensory organs, like cameras or eyes for example. We don't yet completely know how and which neurons, and in what patterns we need to connect them to allow us to produce useful results, but we're getting there, we're reverse engineering the brain [1].

Evolution used billions of years to try out trillions upon trillions of various permutations of chemical setups for each neuron and connections between them... we and other inhabitants of this planet are the result of this vast stochastic optimization, an optimization for a more fit replicator (a gene). We are, as Richard Dawkins noted, that replicator's tools of survival, we are its survival machines [2,3].

In biological organisms born of evolution, there was only one goal, to create a copy (usually mutated due to environmental factors), to create an offspring. Billions and billions of permutations of atoms and simple molecules and environments on this planet eventually resulted in a molecule which was able to copy itself if there was enough of the right material around it to do so. Of course as soon as such a molecule appears in the environment, it quickly consumes all the raw material its able to use to create copies of itself... but due to radiation and the simple fact that biology is not perfect, there are variations of this molecule. Some of these mutant clones of the molecule were smaller and unable to replicate, others were able to do so more efficiently when using raw materials, yet others were even able to break apart surrounding compounds to make the missing necessary raw materials... though it's still just chemistry at this point, in essence this is already competition and predation. The replicating molecules are competing against each other, not by choice, but simply because that's what naturally happens when something can make copies of itself. Anything that does not make a copy, does not take over the environment, and is either expunged from the environment, or used as raw material by replicators.

The molecules split and vary/mutate, new features are added, so that for example some new molecule is able to break apart another molecule, or merge with it. If some complex molecule does not replicate in some manner or another, it has no future... because it will not create an offspring molecule to carry its behavior forward in time.

These mutations, variations, collisions between molecules and atoms, all giving a chance for a more fit replicator to emerge, this was occurring on the entire surface of the planet, and below it. The entire planet was like a computational system, where every inch of the surface gave space for the calculations of the mutations and permutations of molecules to take place... And after billions of years, trillions upon trillions of these replications and mutations, more and more fit systems emerged. Sure, most of the mutations were harmful and produced mostly unfit offspring that could not replicate at all, or were able to replicate but at a slower pace or lower efficiency level... But when you have trillions of opportunities for improvement to work with... no matter how small the probability, eventually, every once in a while... a better combination of molecules results in a better replicator, able to take advantage of some niche within the environment... *That is evolution.*

Through these trillions of permutations, offspring and molecules combined into better replicators, some of which could defend themselves against other replicators, some of which could attack other kinds of replicators so that they could create more of their own kind... To know whom to attack, to know who is composed of the resources that you need to create an offspring, you need a system that is able to tell the difference between the different kinds "stuff" out there, you need some kind of sensory setup... These adaptations continued on and on, and the competition still rages on to this day, from molecules to groups of molecules, cells, the "Survival Machines", tools evolved by the replicators to defend themselves, tools growing more and more complex to deal with other rival replicators and their Survival Machines... a vast biological arms race.

Eventually, through evolution, a new information storage methods was discovered, RNA evolved[9]... the result of all this turmoil is what we see around us today. We are still banding together, we are still competing for limited resources, we are the "Survival Machines" as Dawkins pointed out, machines used by these replicators, by genes, to wage war on each other and make as many copies of themselves as possible. Their newest invention, a feature that evolved to deal with the ever changing and dangerous world, is an interconnected graph of cells that can control these Survival Machines more precisely, deal with much more complex Survival Machines, store information about the world, and keep the genes safe long enough to create more copies of them, with their own survival machine to control. One of the most significant features that arisen in biological organisms, is the parallel biological computer, the vast neural network system, the brain. Over the billions of years of evolution the brain too has been changed, evolution has trended toward more complex brains. Evolution has been slowly exploring the various neural network topologies.

This text is dedicated to the study of evolutionary methods as applied to simulated neural networks. Instead of using atoms and molecules as the building blocks for our evolutionary algorithms, we will use neurons. These neurons, when grouped in particular patterns and topologies, form brains. Biological computers evolved the ability to invent, imagine, scheme, and most importantly, these parallel computers evolved self awareness. Thus we know that such things are possible to evolve, it already happened, nature has proven it possible, we are the proof. In this book we will develop non biological neural networks, and we will apply evolutionary principles to evolve neural systems capable of solving complex problems, adapting to artificial environments, and build a platform that perhaps, some day, could too evolve self aware NN based agents.

In the following section I will discuss in more detail the Biological Neural Networks, how they work, how each neuron processes data, how the neuron encodes data, and how it connects to other neurons in the vast neural network we call our brain.

2.1 Biological Neural Network

Our brain is a vast graph of interconnected neurons, a vast biological neural network. A neuron is just a cell that can accept signals, and based on its chemical and geometrical properties, produce an output. There are roughly 100 billion neurons in the human brain, with trillions of connections between them. Though it might seem surprising that they can work so coherently, the result of which is us, our consciousness and intelligence, it is not surprising at all when we take into account that it took evolution billions of years and trillions of permutations to fine tune this system to get the result that we see today.

A typical neuron, as shown in Fig-2.1, is a cell composed of three main parts, the soma (cell body), the dendrites, and the axon. The soma is a compact body containing the nucleus, and other standard cell internals, and the dendrites and axon are filaments that extrude from it. A single neuron usually has a large number of dendrites, all of which branch profusely but usually retain their filament thickness. Unlike the case with the dendrites, a neuron has only a single axon, originating from a base of the neuron called the “axon hillock”. The axon is usually a long filament which can branch and thus connect to multiple other neurons, with the axonal filament itself usually getting thinner the further it extends and the more it branches. “*Synaptic signals from other neurons are received by the soma and dendrites; signals to other neurons are transmitted by the axon. A typical synapse, then, is a contact between the axon of one neuron and a dendrite or soma of another. Synaptic signals may be excitatory or inhibitory. If the net excitation received by a neuron over a short period of time is large enough, the neuron generates a brief pulse called an action potential, which originates at the soma and propagates rapidly along the axon, activating synapses onto other neurons as it goes.*” [22].

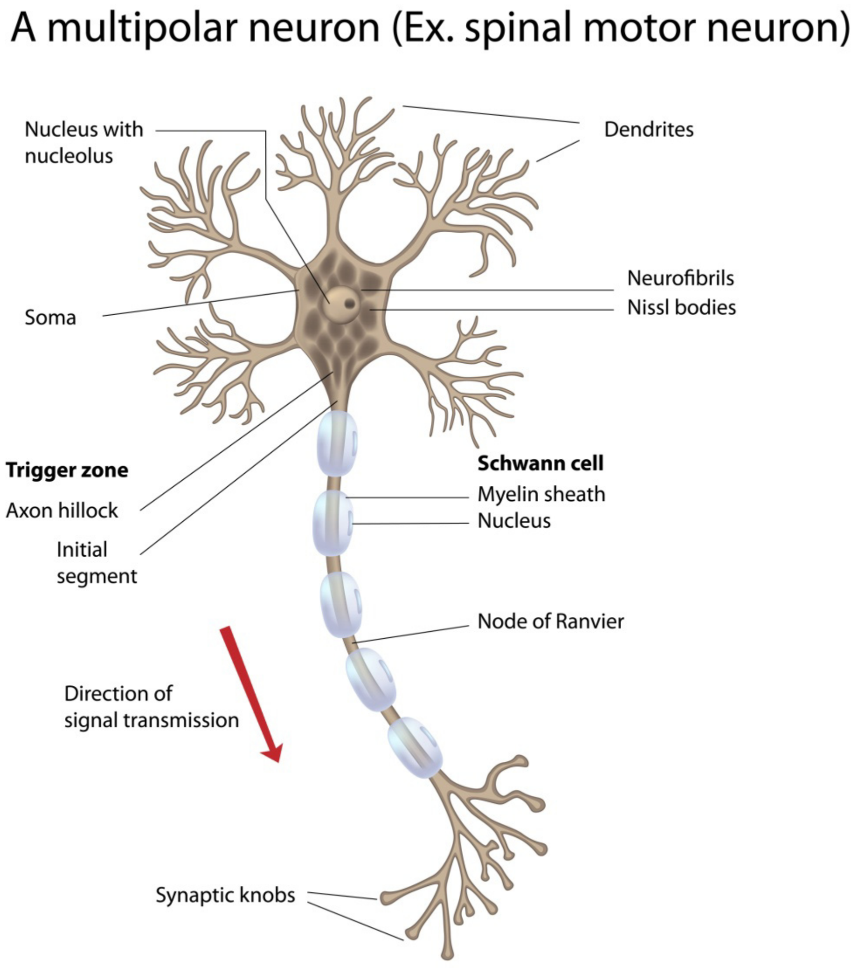


Fig. 2.1 A typical biological neuron.

It would be difficult to describe the biological neuron and its operation any more clearly than is done in the following quote [22] from the ever growing compendium of human knowledge, Wikipedia: "*Neurons are highly specialized for the processing and transmission of cellular signals. Given the diversity of functions performed by neurons in different parts of the nervous system, there is, as expected, a wide variety in the shape, size, and electrochemical properties of neurons. For instance, the soma of a neuron can vary from 4 to 100 micrometers in diameter.*

- *The soma is the central part of the neuron. It contains the nucleus of the cell, and therefore is where most protein synthesis occurs. The nucleus ranges from 3 to 18 micrometers in diameter.*
- *The dendrites of a neuron are cellular extensions with many branches, and metaphorically this overall shape and structure is referred to as a dendritic tree. This is where the majority of input to the neuron occurs.*
- *The axon is a finer, cable-like projection that can extend tens, hundreds, or even tens of thousands of times the diameter of the soma in length. The axon carries nerve signals away from the soma (and also carries some types of information back to it). Many neurons have only one axon, but this axon may—and usually will—undergo extensive branching, enabling communication with many target cells. The part of the axon where it emerges from the soma is called the axon hillock. Besides being an anatomical structure, the axon hillock is also the part of the neuron that has the greatest density of voltage-dependent sodium channels. This makes it the most easily-excited part of the neuron and the spike initiation zone for the axon: in electrophysiological terms it has the most negative action potential threshold. While the axon and axon hillock are generally involved in information outflow, this region can also receive input from other neurons.*
- *The axon terminal contains synapses, specialized structures where neurotransmitter chemicals are released to communicate with target neurons."*

The neuron to neuron signaling is a three step electrochemical process, as shown in Fig-2.2. First an ion based electrical signal is propagated down the axon, and towards every branch of that axon down to the axonal terminals. At the synaptic cleft of those axonal terminals, where the axon is in very close proximity to the cell bodies and dendrites of other neurons, the electrical signal is converted into a chemical one. The neurotransmitters, chemical signals, pass the distance between the axon terminal of the presynaptic neuron, and the dendrite (or soma, and sometimes even axons) of the post-synaptic neuron. How excited the post-synaptic neuron gets, the strength of the signal that the dendrites perceive from these neurotransmitters, all depend on the number of receptors that are present on the surface where the neurotransmitters contact the postsynaptic neuron. Thus, it is the number of, and type of receptors found on the soma and dendrites that weigh the incoming chemical signal, and decide whether it is excitatory when combined with other signals, or inhibitory. The receptors convert the chemical signals they perceive, back into electrical impulses. This train of signals continues its journey down the dendrites and towards the soma. Thus, as we can see, the complete signal is an electrical one, converted into a chemical one, and then converted back into an electrical one.

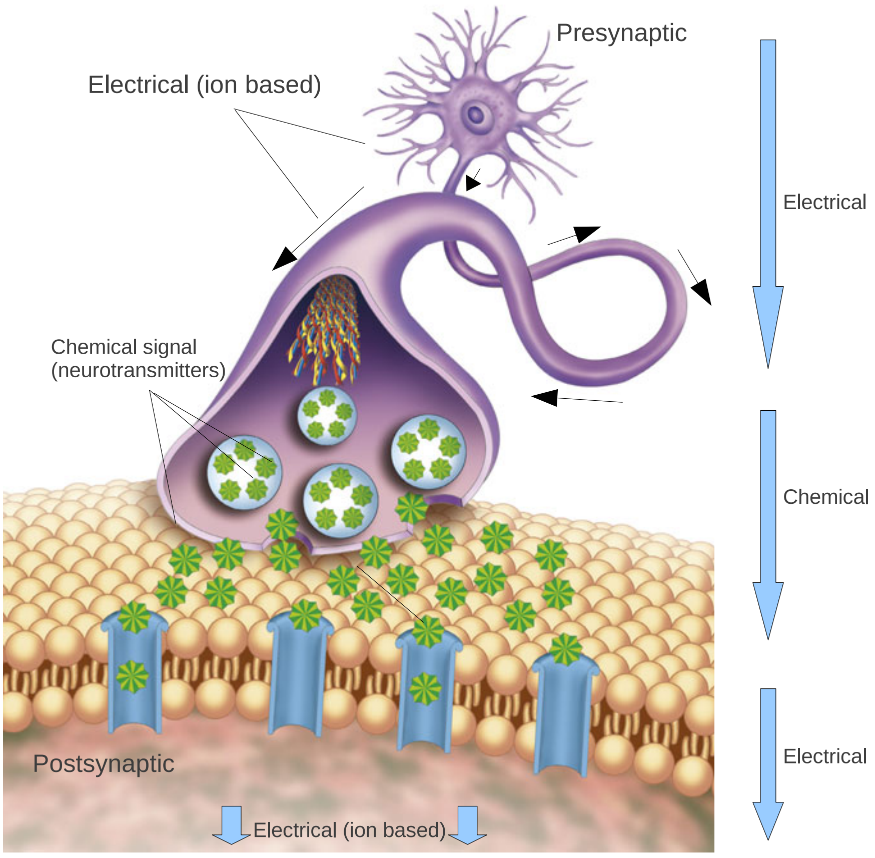


Fig. 2.2 Neuron to neuron signaling, a three step electrochemical process.

Furthermore, the way the signals are perceived is not based on a single spike, a single electrical impulse that some neuron A sends to neuron B, but the signal's frequency. The message is encoded not in the amplitude, but in the frequency. Evolutionary this makes perfect sense, in biological systems it would be difficult to regulate a perfect amplitude as it passes down the wires, but frequency is much simpler to manage using the imperfect biological wetware.

A neuron B could have hundreds to thousands of axons connecting to its soma and dendrites. The way a neuron calculates whether it should produce an output signal, also called action potential or simply spike, at any given time, depends on the intensity of the electrical signal at the axon hillock at that time, as shown in Fig-2.3. Since the intensity of the signal experienced by the axon hillock (trigger zone) depends on how many spikes at that moment excite that region at the same time, the signal is based not only on how many spikes there are, but also on the shape of the neuron and the timing of the signals. The neuron performs a spatio-temporal integration of the incoming signals. If the excitation level at a given time surpasses its threshold, an action potential is generated and passed down the axon. Furthermore, the output signal's amplitude is independent of signals arriving at the axon hillock, it is an all-or-none type of system. The neuron either produces an action potential (if there is enough excitation at the trigger zone), or it does not. Rather than encoding the message in the action potential's amplitude, it is encoded in the frequency, and the frequency depends on the spatiotemporal signal integration and processing that occur within the soma and at the axon hillock.

The signal is based on the spatial properties of the incoming spikes, because if the axon hillock is located in a strange position, or its properties are distributed in space within the neuron differently, it will perceive the incoming signals in a different way. For example, thinking purely mathematically, if the trigger zone is somehow spread thinly over a great area, then to trigger it we would need to send electrical signals that intersect on this wide area, the distribution of the incoming action potentials would have to cover this wide area, all the different places of the axon hillock that sense the electrical signals. On the other hand, if the axon hillock is concentrated at a single point, then to produce the same output we would need to send just a few of the signals towards that point.

On the other hand, the neuron's signal processing is temporal based processing because, if for example 10 spikes come across the axon hillock, each at a rate of 1ms after the other, the axon hillock feels an excitation of only 1 spike every 1ms, which might not be enough excitation beyond the threshold to trigger an output action potential. On the other hand, if 10 spikes come from different sides, and all come across the axon hillock at the same time, the intensity now is 10 spikes rather than one, during the same single ms, which will overcome the biological threshold and the neuron will send an action potential down the axon.

Thus, the output signal, an electrical spike encoded signal produced by the neuron, is based on the spatial and temporal properties of its input signals. Something similar is shown in Fig-2.3, where I loosely defined the timings of when the spikes will arrive at the trigger zone using t which defines the arrival at the trigger zone, t-1 which defines arrival at the trigger zone in 1 delta, t-2 which defines the arrival at the trigger zone in 2 deltas, and so on. At *t-1* we see that there will be 4 spikes, at *t-2* only 2. If it requires 3 spikes to overcome the threshold (which itself is defined by the shape and chemical properties at the axon hillock) and to set off an action potential down the axon, then the signals arriving at *t-2*, when they do finally arrive at the hillock in 2 deltas (time units), will not trigger an action potential, while the signals currently at *t-1* will generate a spike when they finally arrive at the trigger zone.

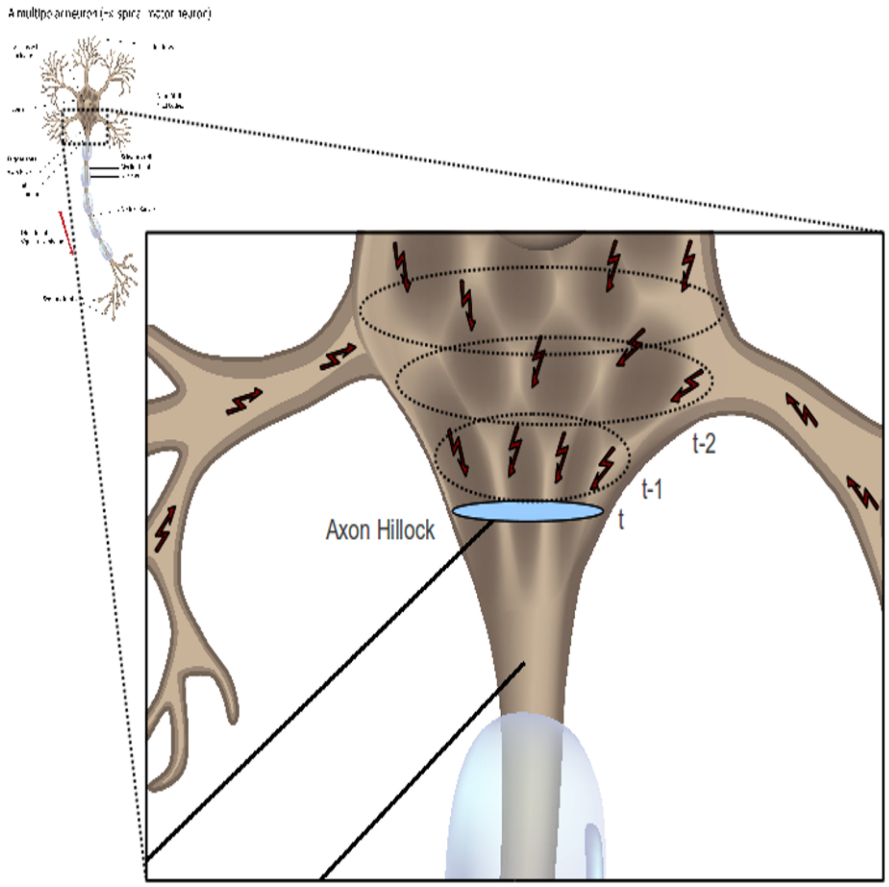


Fig. 2.3 Spatiotemporal signal integration.

Furthermore, the neurons don't just accept incoming signals and produce outgoing signals, the neurons also change over time based on the signals they process. This change in the way neurons respond to signals by adding more receptors to the dendrites, or subtracting receptors from the dendrites, or modifying the way their receptors work, is one of the processes by which a neural network learns and changes its excitability towards certain signals, it is how we accumulate experience and form memories. Other ways by which a neural network learns is through the axons branching and making new connections, or breaking old connections. And finally the NN changes in the way it processes signals through having the very fluid in which the neurons are bathed changed and chemically modified, through drugs or other means for example.

The most important part to take away from this chapter is that the biological neurons output frequency encoded signals, and that they process the incoming frequency encoded signals through spatiotemporal integration of those signals. And that the neurons can change over time based on the signals they process, the neurons change biologically, they change their information processing strategies, and they can form new connections to other neurons, and break old ones. This process is called *neuronal plasticity*, or just plasticity. In the next section we will discuss artificial neural networks, how they function, and how they can differ from their biological counterparts.

2.2 Artificial Neural Network

Artificial neural networks (NN), as shown in Fig-2.4, are simulated biological neural networks to different levels of precision. In this section we will cover the typical artificial neural network, which are not perfect simulations. A typical artificial neuron, aka neurode, *does not* simulate a biological neuron at the atomic, or even molecular level. Artificial neurons are abstractions of biological neurons, they represent the essentials of biological neurons, their nonlinear signal integration, plasticity, and concurrency.

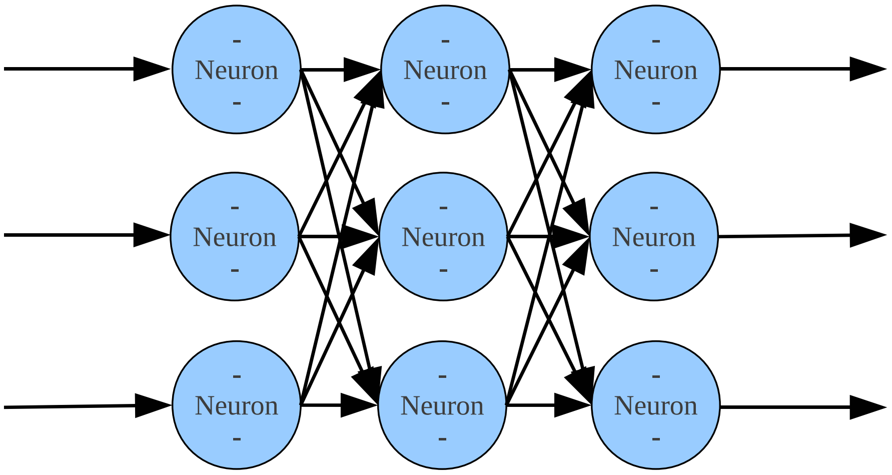


Fig. 2.4 An artificial neural network.

As shown in Fig-2.5, like a biological neuron, an artificial one accepts signals through its artificial dendrites, processes those signals in its artificial soma, and outputs the processed signals to other neurons it is connected with. It is a concise representation of what a biological neuron does. A biological neuron simply accepts signals, weighs each signal, where the weight depends on the receptors on the dendrites on which the axons from other neurons intercepted, then based on its internal structure and chemical composition, produces the final frequency encoded output and passes that output onwards to other neurons. In the same way, an artificial neuron accepts signals, weighs each signal using its weight parameters, integrates all the weighted signals through its activation function which simulates the biological neuron's spatiotemporal processing at the axon hillock, and then propagates the final output signal to other neurons it is connected to.

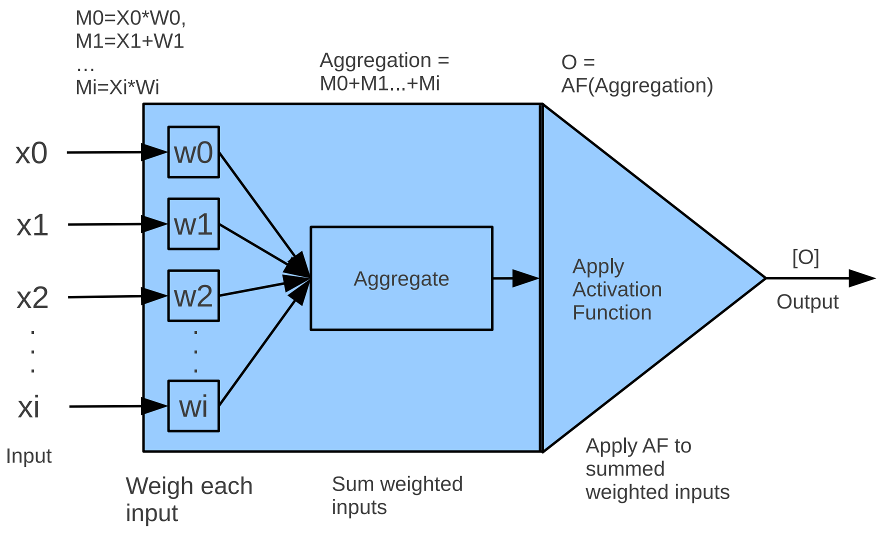


Fig. 2.5 A detailed look at an artificial neuron's schematic.

As can be seen from Fig-2.5, there are of course differences. We abstract the functionality undertaken by the receptors on the dendrites with simple weights, nevertheless, each incoming signal is weighted, and depending on whether the weight is positive or negative, each incoming signal can act as an excitatory or inhibitory one, respectively. We abstract spatiotemporal signal integration that occurs at the axon hillock with an activation function (which can be anything, and as complex as the researcher desires), nevertheless, the weighted signals are integrated at the output point of the artificial neuron to produce the final output vector, which is then passed onwards to other neurons. And finally, we abstract the functionality undertaken by the axon with simple signal message passing, nevertheless, the final output signal is propagated, diligently, to all postsynaptic artificial neurons.

The biological neural network is a vast graph of parallel processing simple biological signal integrators, and the artificial neural network too is a vast graph of parallel processing simple signal integrators. The neurons in a biological neural network can adapt, and change its functionality over time, which too can be done in artificial neural network through simulated neural plasticity, as we will discuss in later sections, and eventually implement in the NN systems we will build ourselves.

There is one thing though that differs significantly in the typical artificial neural networks, and the biological neural networks. The neurons in a biological NN frequency encode their signals, whereas in the artificial NNs, the neurons amplitude encode their signals. What has more flexibility? Frequency encoded NN systems or the amplitude encoded ones? It is difficult to say, but we do know that both, biological and artificial neural networks are Turing complete [4], which means that both possess the same amount of flexibility. The implications of the fact that both systems are universal Turing machines is that even if a single artificial neuron does not do as much, or perform as a complex computation as a single biological neuron, we could put a few artificial neurons together into an artificial neural circuit, and this artificial neural circuit will have the same processing power and flexibility as a biological neuron. On the other hand, note that frequency encoding signals takes more time, because it will at least take the amount of time between multiple spikes in the spike train of the signal for the message to be forwarded (since it is the frequency, the time between the spikes that is important), whereas in an amplitude encoded message, the single spike, its amplitude, carries all the information needed.

How much of the biology and chemistry of the biological neuron is actually needed? After all, the biological neuron is the way it is now due to the fact that it was the first *randomly found solution*, the easiest solution found by evolution. Wetware has no choice but to use ions instead of electrons for electrical signal propagation. Wetware has no choice but to use frequency encoding, instead of amplitude encoding, because wetware is so much more unreliable than hardware (but the biological neural network as a whole, due to a high level of interconnections, is highly fault tolerant, reliable, and precise). The human neuron is not a perfect processing element, it is simply the processing element that was found through evolution, by chance, the easiest one to evolve over time, that's all. Thus, perhaps a typical plasticity incorporating artificial neuron has all the right features already. We have after all evolved ALife organisms with just a few dozen neurons that exhibited interesting and evolutionary appropriate behaviors with regards to food foraging and hunting [5,6,7]. We do know one thing though, the limits of speed, signal propagation, neural plasticity, life span of the neuron, integration of new neural systems over the organism's lifetime, are all limited in wetware by biology. *None of these limitations are present in hardware*, the only speed limit of signal propagation is that of light in a hardware based neural computing system. The non biological neural computer can add new neural circuits to itself over lifetime, and that lifetime span is unlimited, given that hardware upkeep is possible.

I think that amplitude encoded signaling is just as powerful, and the activation functions of the artificial neurons, the integration of the weighted signals, is also as flexible, or can be as flexible as the spatiotemporal signal integration performed by a biological neuron. An artificial neuron can simulate different kinds of receptor densities on the dendrites by different values for weights. An artificial neuron can simulate different kinds of neuron types through the use of different kinds of activation functions. Even plasticity is easy to add to an artificial neuron. And of course, there are also artificial spiking neural network systems [23,24,25], which use frequency encoding like a biological neural network does. There is absolutely no reason why artificial neural networks cannot achieve the same level of performance, robustness, and intelligence, as biological neural networks have.

2.2.1 The Neurode in Detail

In this section we will do a more detailed analysis of the architecture of an artificial neuron, how it processes an incoming signal, and how such an artificial neuron could be represented in software. In Fig-2.6 we use the schematic of an artificial neuron in a simple example where the neuron receives two incoming signals. Each of the signals is a vector. The third signal is not from any other neuron, but is simply a bias value, which modifies the neuron's processing. The neuron processes the signal based on its internals, and then forwards its output, in a vector form, to postsynaptic neurons. In the figure, the "axon" of the neuron branches into 3 strands.

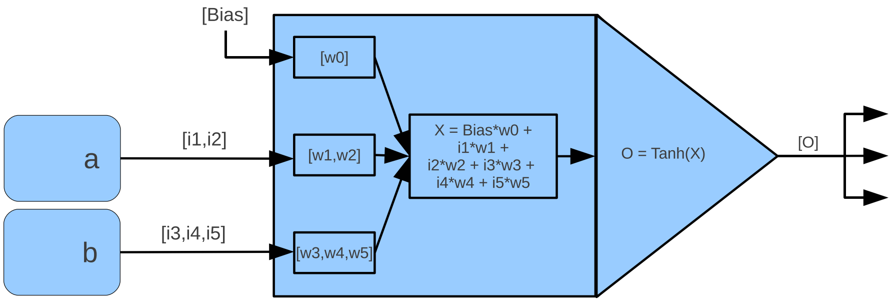


Fig. 2.6 An artificial neuron in action, receiving signals from two other elements, a and b.

Artificial neurons accept vector input signals, and output a vector signal of length 1. Each input signal is weighted; each element in the input vector is multiplied by a weight in a weight list associated with that input vector, and that particular element in the input vector. Thus, the integration of the incoming signals is done by calculating a dot product of the incoming vectors and the weight vectors associated with those input vectors. In the above figure, there are two incoming signals from other elements, and a bias signal (which we'll talk about next). The incoming signal from element 'a' is a vector signal of length 2, the signal from element 'b', is a vector of length 3, and the bias signal is a vector of length 1. The neuron has a weight list for each incoming signal. The weight lists weigh the importance of each input vector. The way we integrate the input signal is by calculating a dot product of the weights and the input signals. Once the dot product is calculated, we compute the output of the neuron, Output = F(X), where F is the activation function, and X = Dot_Product + Bias. The neuron then packages this result into a vector of length 1, like so: [Output], and then fans out this output vector to the elements that it is connected to. A sigmoid function, or hyperbolic tangent, is the typically used activation function in artificial neurons. A multi-layered feed forward neural circuit composed of neurons using sigmoid activation functions can act as a universal function approximator [8], which means that a neural network composed of such neural circuits can do anything.

Now regarding the bias input, it is simply an input vector which is used to increase the flexibility of the neuron by giving it an extra weight that it can use to skew the dot product of the input signals. Not every neuron needs to have a bias input, it's optional, and if the weight for the bias input is 0, then that is equivalent to a neuron that does not have a bias input at all. The neuron can use the bias to modify the point at which the weighted dot product produces a positive output when passed through the activation function, in which case the bias acts as a threshold. If the bias is a large positive number, then no matter what the input will be, the neuron has a much greater chance of outputting a positive value. If the bias is a negative number, then the incoming signals will have to be high enough to overcome this bias for the neuron to output a positive value. In essence, the bias controls how excitable in general the neuron is, whereas the weights of the non bias inputs control how significant those inputs are, and whether the neuron considers them excitatory or inhibitory. In future figures we will use a much simpler neuron schematic than the one we used in Fig-2.6. Having now demonstrated the inner workings of a neuron, in the future when diagramming a neuron we will use a circle, with multiple inputs, and an output link that fans out the neuron's output signal.

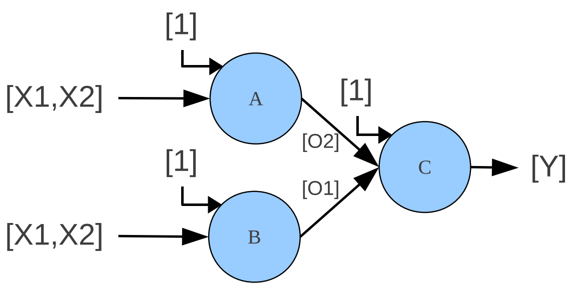
When we connect a few of these neurons together in the right topology and set their weights to the right values, forming a small neural network like the one in Fig-2.7, such a neural network could perform useful tasks. In Fig-2.7 for example, the neural circuit composed of 3 neurons calculates the XOR of the inputs. We can demonstrate that this neural circuit does indeed calculate the XOR of its inputs by feeding it the signals from a XOR truth table, and comparing its output to the proper output of the XOR logical operator. The input signals, in this case a single vector of length 2, is fed from the truth table to the neurons A and B, each neuron calculates an output signal based on its weights, and then forwards that signal to neuron C. Then neuron C calculates an output based on the inputs it receives from neuron A and B, and then forwards that output onwards. It is this final output, the output of the neuron C, that is the output of the neural circuit. And it is this output that we will compare to the proper output that a XOR logical operator would produce

Table 1. The XOR truth table, and the vector form which can be used as input/output signals of a NN. In this table, 1 == true, -1 == false.

Pattern	**[X1, X2, Y]**	**Input: [X1, X2]**	**Output: [Y]**
1	[-1,-1,-1]	[-1,-1]	[-1]
2	[-1, 1, 1]	[-1, 1]	[1]
3	[1,-1, 1]	[1,-1]	[1]
4	[1, 1,-1]	[1, 1]	[-1]

when fed the same input signals as the neural circuit at hand. The XOR truth table is shown in the following table, where X1 and X2 are the inputs to the XOR logical operator, and Y is the XOR operator's output.

We will now walk through the neural circuit, neuron by neuron, step by step, and calculate its output for every input in the XOR truth table. As shown in Fig-2.7, the neural circuit has 3 neurons, A, B, and C. Neuron A has the following weight vector: [2.1081,2.2440,2.2533], where: W1=2.1081, W2=2.2440, and Bias=2.2533. Neuron B has the following weight vector: [3.4963,-2.7463,3.5200], where W1=3.4963, W2=-2.7463, and Bias = 3.5200. Finally, Neuron C has the following weight vector: [-2.5983,2.7354,2.7255], where W1=-2.5983, W2=2.7354, and Bias=2.7255. With this information we can now calculate the output of the neural circuit for every input vector, as shown in Fig-2.7.



Input [X1,X2]: [-1,-1]
A: O1 = -0.9704 = tanh(-1*2.1081+ -1*2.2440 + 1*2.2533)
B: O2 = 0.9922= tanh(-1*3.4964 + -1*-2.7464 + 1*3.5200)
C: Y = **-0.99** = tanh(0.9922*-2.5983 + -0.9704*2.7354 + 1*2.7255)

Input [X1,X2]: [-1,1]
A: O1 =0.9833 = tanh(-1*2.1081+ 1*2.2440 + 1*2.2533)
B: O2 = -0.9914= tanh(-1*3.4964 + 1*-2.7464 + 1*3.5200)
C: Y = **0.99** = tanh(-0.9914*-2.5983 + 0.9833*2.7354 + 1*2.7255)

Input [X1,X2]: [1,-1]
... C: Y = **0.99**

Input [X1,X2]: [1,1]
... C: Y = **-0.99**

Fig. 2.7 Calculating the output of the XOR neural circuit.

As can be seen in the above figure, the neural circuit simulates a XOR. In this manner we could even build a universal Turing machine, by combining such XOR neural circuits. Another network of neurons with another set of activation functions and neural weights would yield something different...

The main question though is, how do we figure out the synaptic weights and the NN topologies needed to solve some problem, how for example did we figure out the weights for each of these 3 neurons to get this neural circuit to act as a XOR operator? The answer is, a learning algorithm, an automated algorithm that sets up the weights. There are many types of algorithms that can be used to setup the synaptic weights within a NN. Some require that we have some kind of training sample first, a set of inputs and outputs, which a mathematical function can then use to set up the weights of a neural network. Other algorithms do not require such prior knowledge, all that is needed is for each NN to be gaged on how well it performed and how its performance on some problem compares to those of other NNs. We will discuss the various learning algorithms in section 2.4, but before we move on to that section, we will first cover the standard Neural Network terminology when it comes to NN topological structures, and discuss the two types of basic NN topologies, feedforward and recurrent, in the next section.

2.3 Neural Networks and Neural Network Based Systems

A neuron by itself is a simple processing element. It is when we interconnect these neurons together, in parallel and in series, when we form a neural network (NN), that true computational power emerges. A NN is usually composed of multiple layers, as the example shows in Fig. 2.8. The depth of a NN is the number of layers that compose it.

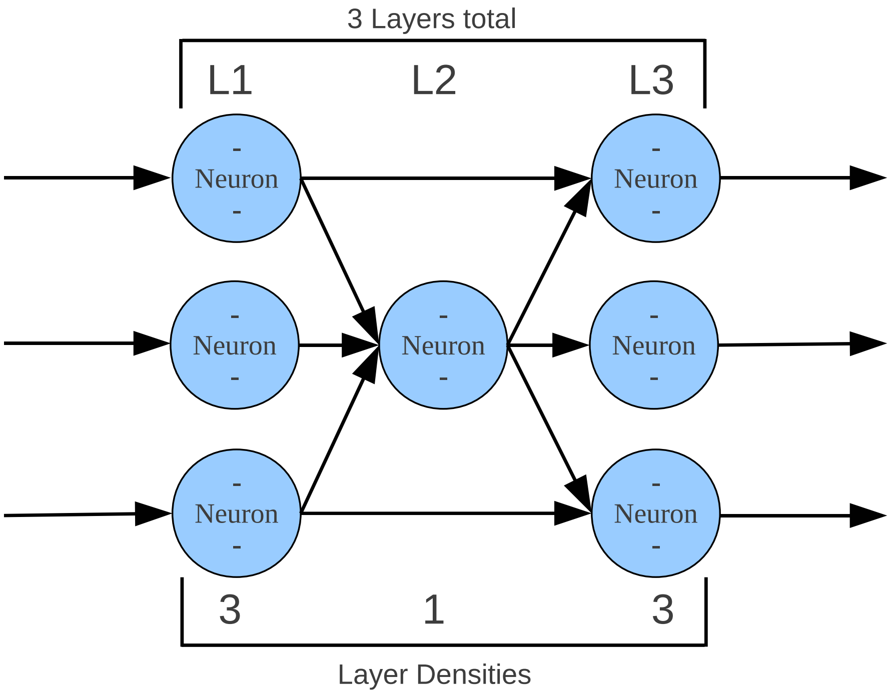


Fig. 2.8 A multi-layered NN, with a NN composed of 3 layers. The first layer has 3 neurons, the second layer has 1 neuron, and the third layer has 3 neurons.

Using layers when discussing and developing NN topologies gives us an ability to see the depth of a NN, it gives us the ability to calculate the minimum number of neurons the input has to be processed by in series, before a NN produces an output. The depth tells us the minimum amount of non parallel processing that has to be done by a distributed NN. Finally, assigning each neuron a layer allows us to see whether the connections from one neuron to another are feed forward, meaning some neuron A sends signals to a neuron B which is in front of neuron A, or whether the connection is recurrent, meaning some neuron A sends a signal to neuron B which itself is behind A, and whose original output signal is either fed directly to neuron A, or was forwarded to other neurons and then eventually got to neuron A before it itself produced its output signal (the recurrent signal that it sent back to neuron B). Indeed in recurrent NNs, one can have feedforward and feedback loop based neural circuits, and a neuron B could have sent a signal to neuron A, which then processed it and sent its output back to neuron B... When a neural network is composed of neurons whose output signals go only in the forward facing direction, such a network is called a feedforward NN. If the NN also includes some recurrent connections, then it is a recurrent NN. An example of a feedforward and a recurrent neural network is shown in Fig-2.9.

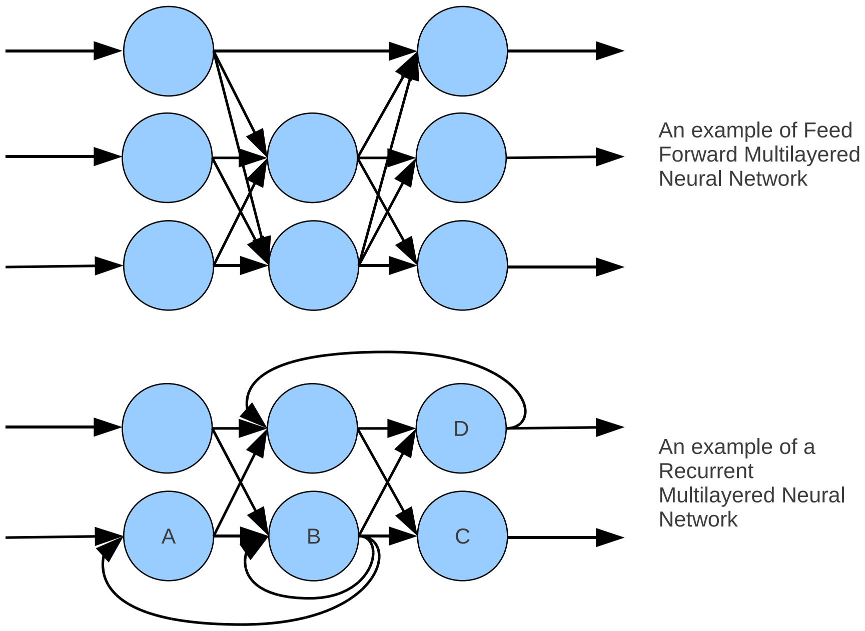

An example of Feed Forward Multilayered Neural Network
D
A
B
C
An example of a Recurrent Multilayered Neural Network

Fig. 2.9 An example of a Feedforward and a Recurrent neural network.

As can be seen in the recurrent NN example, neuron A receives a signal from somewhere, processes it, sends a signal to neuron B, which processes the signals sent to it and then sends an output signal to neuron C, D, but also a recurrent signal back to A and itself.

2.3.1 Recurrent Neural Networks and Memory Loops

What is significant about recurrent neural networks is that they can form memory circuits. For example, the Fig-2.10 shows four examples of a recurrent NN. Note that in 2.10A, the neuron sends a signal back to itself. This means that at every moment, it is aware of its previous output, and that output is taken into account when producing a new output. The neuron has memory of its previous action, and depending on the weight for that recurrent connection, its previous signal at time step T can play a large or a small part in its output at a time step T+1. In 2.10C neuron *1* has a recurrent connection to neuron *2*, which outputs a signal back to neuron *1*. This neural circuit too forms a memory system, because this circuit does not simply process signals, but takes into account the information from time step T-2, when making a decision with regards to the output at time step T. Why T-2?, because at T-2 neuron *1* outputs a signal to *2* rather than itself, it is

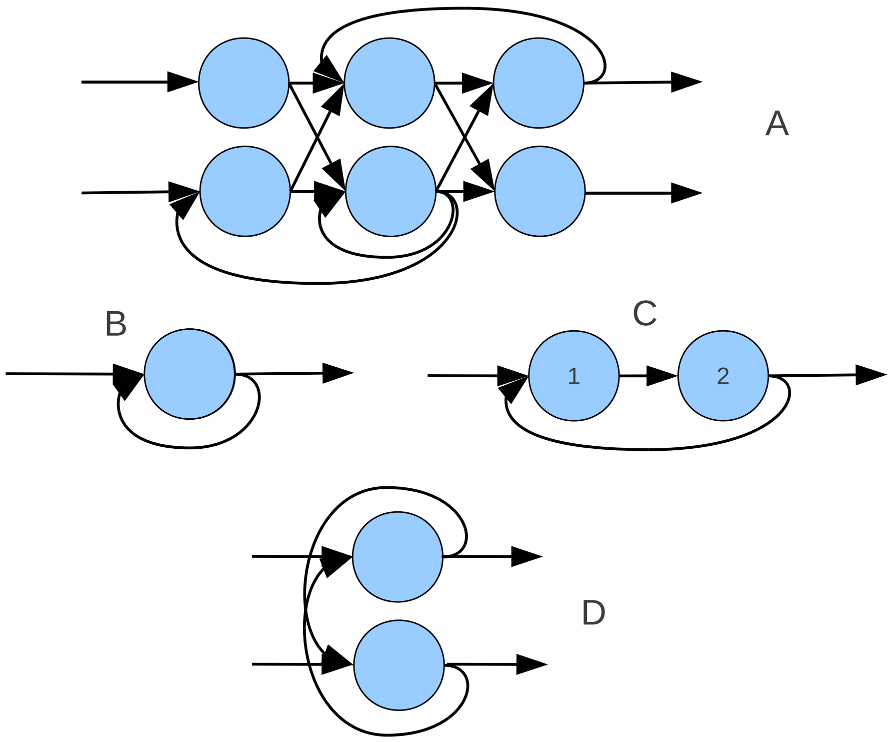


Fig. 2.10 An example of recurrent NNs that could potentially represent memory loops. A is a general, 3 layer recurrent neural network, with 3 recurrent connections. B is a self recurrent neuron, which thus has a trailing memory of its previous output, depending on its weight with its own recurrent connection. C is a two layer recurrent NN, with neuron-2 receiving a signal from neuron-1, which processes the signal that came from neuron-2 in the first place, thus neuron-2 receives a signal that it itself produced T-2 steps before, processed by neuron-1. Finally, D is a one layer recurrent NN, which has the topology of a flip flop circuit.

then at T-1 that *2* outputs a signal to *1*, and it is only at time T that *1* outputs a signal after processing an input from some other element, and a signal it output at T-2, which was processed by *2* before coming back to *1* again. Thus this memory loop is deeper, and more involved. Even more complex systems can of course be easily evolved, or engineered by hand.

2.3.2 A Neural Network Based System

We have discussed neural networks, and in all figures I've shown the NNs as having input signals sent to them from the outside, but from where? In real implementations the NNs have to interact with the real or simulated world, and the signals they produce need to be somehow used to accomplish useful tasks and act upon those real or simulated worlds. For example, our own brain accepts signals from the outside world, and signals from our own body through the various sensory organs, and the embedded sensory neurons within those organs. For example our eyes, our skin, our nose... are all sensory organs with large concentrations of sensory elements that feed the signals to the vast neural network we call our brain. These sensory organs, these sensors, encode the signals in a form that can be forwarded to, and understood by, the brain.

The output signals produced by our brains also have no action without some actuators to interpret those signals, and then use those signals to act upon the world. The output signals are made sense of by the actuators, our muscles for example evolved to know how to respond when receiving signals from the motor neurons, and it is our muscles that perform actions upon the world based on the signals coming from the biological NN.

Thus, though it is the NN that thinks, it is the NN with sensors and actuators that forms the whole system. Without our sensory organs, our brain is in the dark, and without our muscles, it does not matter what we think, because we can have no affect on, and no way to interact with, the world.

It is the same with artificial NNs. They require sensors, and actuators. A sensor can be a camera, which can package its output signals in a way that can be understood by the NN, for example by representing the sensory signals as vectors. An actuator can be a motor, with a function that can translate the NN's output vector into electrical signals that controls the actual motor.

Thus it is the whole thing, the sensors connected to and sending the sensory signals to the NN, and the NN connected to and sending its output signals to the actuators, that forms the full system, as shown in Fig-2.11. In this book we will refer to such a complete and self contained system, the Sensors connected to the Neural Network, which itself is connected to Actuators, as the NN based system, or NN based agent. It is only when we are discussing the NN in isolation, the topology of a NN for example, that I will use the term NN on its own. When it's clear from the discussion though, the two terms will sometimes be used interchangeably.

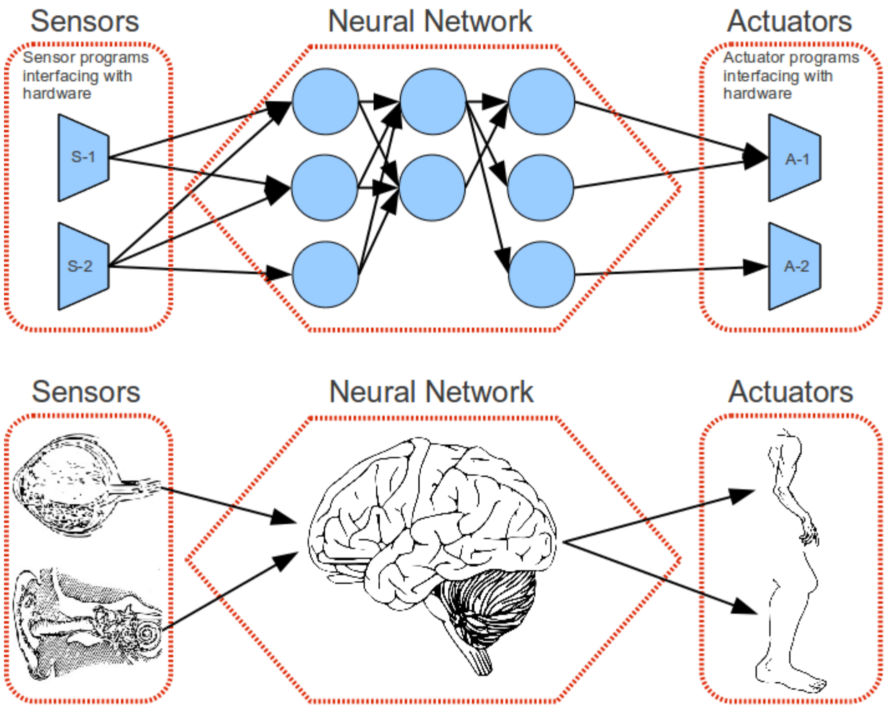

Sensors
Neural Network
Actuators
Sensor programs interfacing with hardware
Actuator programs interfacing with hardware
S-1
S-2
A-1
A-2
Sensors
Neural Network
Actuators

Fig. 2.11 The Biological and the Artificial Neural Network Systems compared.

Having now discussed the basics of NNs, the different types of topologies, and what a complete NN system is, and what parts form a NN system, we now move forward and briefly cover how the classical, typical NNs learn and get trained. In the following sections we will discuss the typical algorithms used to modify the weights of the neurons belonging to some NN applied to a problem, and the difference between the term learning and training.

2.4 Learning Vs. Training

Though most of the time you will hear the terms *learning* and *training* used interchangeably when people discuss the processes and algorithms that modify the weights and the topology of a NN such that it is more fit, such that it is able to solve some problem it is applied to, in this book we will discriminate between the two. Take for example the Back Propagation (BP) Learning algorithm we will discuss in the next section. In that algorithm we have a list of tuples of inputs and expected outputs. The inputs are the vectors we would feed to a NN system, and the outputs are the expected outputs we'd like the NN system to produce. The way the BP learning algorithm works is by letting a neural network output a vector based on the input, and then use a gradient descent method to change the weight parameters of the neurons based on the difference of the NN's actual output, and the expected output. Through the application of the BP algorithm, eventually the difference between the NN's output and the expected output, is minimized. Once the error, the difference between the NN's output and the expected output is below some threshold, we apply this NN to data that it has not yet seen, and use the NN's output as the result, hoping that the NN can generalize from the training set to this new real world data. When using this algorithm, is the NN really learning?

When we think of learning, we think of studying, of one looking at the data, and then through logic, and explanation to oneself, coming to a conclusion that something should work this way or that way. The starting NN, and the end result, are the same NN, and the change to the reasoning, and thus to the topology and synaptic weights is self initiated and self inflicted. We are the same before and after we learn something, in a sense that this change in our logic in our perception was not done from the outside by some external system, but instead, it was us that has done the change, it was us that had worked and came to the conclusion that another way of thinking is better, or that something works this particular way... That is learning. In the BP algorithm we just discussed above, the NNs are static, they are not learning. We simply bring into existence a NN, see whether how it behaves now is appropriate and whether it represents the answer to some question, and then we change its weights, the synaptic weights of the neurons are modified and optimized from the outside, by an outside supervisor. The NN is trained. This is something that is referred to in the standard Neural Network literature as Supervised Learning, where the NN has a supervisor that tells it whether its answers are right or wrong, and it is the supervisor (an external algorithm) that modifies the NN so that the next time it will hopefully produce a better answer.

In true learning, the NNs are able to change on their own through experience. The NN only lives ones, and during that lifetime it is modified through experience. And what experience it is exposed to is to a great degree guided by the NN itself. In the way that what we choose to expose ourselves to, influences what we learn, and how our perspectives, how we think, and what we know, changes. The phenomenon of the neural networks changing and adapting through experience, is due to neural plasticity. Neural plasticity is the ability of the neuron to change due to experience. Thus for example if we create a large NN system composed of plastic (those possessing plasticity) neurons, and then release it into a virtual environment and it improves on its behavior, it learns how to survive in the environment through experience... that is what I would refer to as learning. This is called Unsupervised Learning, and indeed that is completely possible to do in artificial neural networks, by for example giving each neurode the functionality which allows it to change its information processing strategy based on the signals it processes.

Thus the main idea to be taken from this section with regards to the difference between what I call training and learning, is this: The process of training a neural network is accomplished by changing its weights and topology from the outside, by some algorithm external to the NN based system. On the other hand, a neural network is learning if it is adjusting and improving itself of its own volition, through its exposure to experience and the change of its NN topology and neural parameters. Thus it would be possible to bootstrap a NN system, by first training some static system, then adding plasticity to the NN, and then releasing this bootstrapped NN system into some environment, where based on the bootstrapped part it is able to survive, and as it survives it is being exposed to the environment, at which point its plastic neural system changes and adapts, and the NN learns. We will explore this further in later chapters, after we’ve built a neuroevolutionary system that can evolve NN systems, and where the NN systems are then released into some simulated environment. We will evolve NN systems which have plasticity, we will evolve them so that they can use that plasticity to *learn* new things on their own.

In the following two sections we will discuss the typical supervised and unsupervised training and learning algorithms respectively.

2.5 Neural Network Supervised “Learning” Algorithms

Supervised learning is a machine learning approach to inferring a target function from a training data set composed of a set of training examples. Each training example is composed of an input vector, and a desired or expected output vector. The desired output vector is also referred to as the supervisory signal. When applied to neural networks, supervised learning, or training, is an approach to the modification and automation of weight setting of a neural network through the use of a supervisor, or external system, that compares the NN’s output to a correct, pre-calculated output, and thus expected output, and then based on the difference between the NN’s output and the expected output, modifies the weights of the neurons in the NN based on some optimization algorithm. A supervised “learning” algorithm can only be applied to problems where you already know the answers, where you can build a *training set*. A training set is a list of tuples, where every tuple is composed of the input vector, and the expected output vector: [{Input, ExpectedOutput}...]. Thus we need to know the outputs ahead of time, so that we can train the neural network before we can use it with input signals it has not yet seen. Note, this is not always possible. For example, let’s say we wish to create a neurocontroller for a robot, to survive in some environment. There is no training set for such a problem, there is no list of tuples where for every camera input that act as robots eyes there is an expected and correct move that the robot must make.

That is usually never the case, in fact, we do not know what the right move is, if we knew that, we would not need to create the neurocontroller. Another example is the creation of a neurocontroller that can make a robotic arm reach for some point in space. Again, if we knew what the right combination of moves that the motors needed to make, we would not need for the NN to figure that out.

The most widely used of such supervised algorithms, is the Error Backpropaga–tion algorithm [10]. The backpropagation algorithm uses gradient descent to look for the minimum error function between the NN's output, and the expected output. The most typical NN topology that this algorithm is applied to, is a standard feed-forward neural network (though there is a BP algorithm for a recurrent NN topology too). As we discussed, a supervised learning algorithm trains a NN to approximate some function implicitly, by training the NN on a set of inputs and expected outputs. The error that must be minimized is the error between the NN's output, and the expected output.

Because we will concentrate on neuroevolution, we will not cover this algorithm in great detail. But an extensive coverage of this supervised learning algorithm can be found in: [11,12]. In summary, the training of the NN through the backprop algorithm works as follows:

1. Create a multi-layered feed forward neural network, where each neuron has a random set of weights. Set the neurons in the first/input layer to have X number of weights, plus bias, where X is the vector length of the input vectors. Set the last/output layer to have Y number of neurons, where Y is the length of the expected output vector.
2. **For every tuple(i) in the training list, DO:**
   3. Feedforward Phase:
      1. Forward Input(i) vector to the neurons in the first layer of NN.
      2. Gather the output signals from the neurons in the last layer of NN.
      3. Combine the gathered signals into an Output(i) vector.

   4. Backprop Phase:
      1. Calculate the error between the NN's Output(i) and ExpectedOutput(i)
      2. Propagate the errors back to the neurons, and update the weights of the neurons based on their contribution to that error. The weights are updated through gradient descent such that the error is decreased.
      3. The errors are propagated recurrently from the last neural layer to the first.

5. **EndDO**
6. Repeat steps 2-5 until the average total error between the NN's outputs and the expected outputs is less than some chosen value *e*.

Schematically, the feedforward phase and the error backprop phase, is demonstrated in Fig. 2.12.

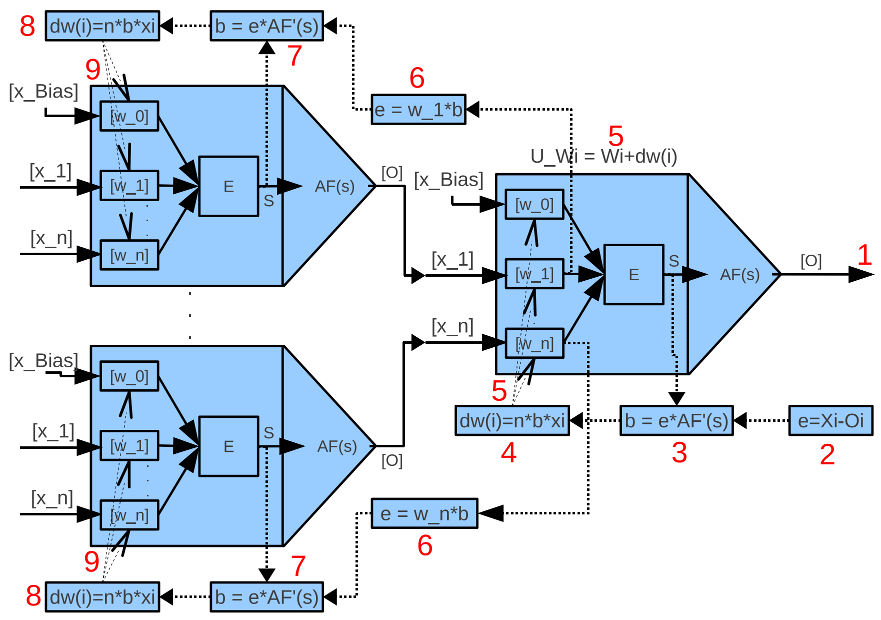


Fig. 2.12 The schematic of the backprop learning algorithm.

The steps of recursively updating the synaptic weights of all the neurons in the feedforward NN based on the error between the NN's output and the expected output is demonstrated by the figure through the step numbers. Starting with step *1*, the NN's output is *O*, and the expected output is *X*. If there were more than one output neurons, then each neuron *i* would produce an output *Oi*, and for each *Oi* there would be an expected output *Xi*. The meaning of the steps is elaborated on in the following list:

1. The neuron in the output layer of the feedforward NN produces an output O.
2. The error of the neuron's output as compared to the expected output is e, calculated as: *e = Xi-Oi* where Xi and Oi are the output of neuron i, and expected output i, respectively, if there are i number of output neurons in the NN.
3. We calculate b (beta) by multiplying the derivative of the activation function by e: *b = e*AF'(S)*, where *S* is the dot product of the neuron's input signals and synaptic weights for those input signals.
4. We then calculate the delta (change in) weight for each weight i as follows: *dw(i) = n*b*Xi,* where n is a learning parameter chosen by the researcher (usually between 0.01 and 1), b is the value calculated in step-3, and Xi is the input i to the neuron, associated with the weight i.
5. We updated every synaptic weight i of the neuron using the appropriate *dw(i)* for each *Wi*. The updated weight is produced through: *U_Wi = Wi+dw(i)*.
6. The next *e (error)* is recursively calculated for every presynaptic neuron using the equation: *e=Wi*b*, where Wi is the synaptic weight associated with the neuron whose output was Xi.

7. We calculate b (beta) by multiplying the derivative of the neuron's activation function by e: *b = e*AF'(s)*.
8. We then calculate delta weight for each weight i as follows: *dw(i) = n*b*Xi* where n is a learning parameter chosen by the researcher (usually between 0.01 and 1), b is the value calculated in step-7, and Xi is the input i to the neuron, associated with the weight i.
9. We updated every synaptic weight i of the neuron using the appropriate dw(i). The updated weights of the neurons are calculated through: *U_Wi = Wi+dw(i)*.

This procedure is continued recursively to the other presynaptic neurons, all the way to, and including, the first layer neurons.

Thus, to optimize the neural network's weights for some particular task for which we have a training set, we would apply the backprop algorithm to the NN, running it through the training set multiple times, until the total error between the NN's output and the expected output is low enough that we consider the NN's synaptic weights a solution. At this point we would apply the NN to the real problem for which we have been training it.

As noted, this can only be applied to the problems for which we already know the answers, or a sample of answers. This algorithm is used only to train the neural network, once it is trained, its weights will remain static, and the neural circuit is used as a program, unchanging for the remainder of its life. There are numerous extensions and improvements to this basic algorithm, covered in the referenced texts. But no matter the improvements, at the end of the day it is still a supervised approach, and the resulting NN is static. In the next section we will briefly discuss unsupervised learning algorithms, the addition of plasticity to the neurons of a NN, and other methods which allow the NN to self organize, and adapt and change through the interaction with the environment, and/or data it comes across.

2.6 Neural Network Unsupervised Learning Algorithms

Unsupervised learning refers to the problem of trying to determine structure in incoming, unlabeled data. In such a learning algorithm, because the input is unlabeled, unlike the case with the training data set discussed in section 2.5, here there is no error or reward signals which can be used to guide the modification process of neural weights based on the difference between the output and the expected output. Instead, the NN self modifies its parameters based on the inputs and its own outputs through some algorithm. There are two general kinds of such learning algorithms; a learning algorithm can either be a system that has a global view of the NN, and which uses this global view to modify neural weights (kohonen, competitive…), or a learning algorithm can be a local on, embedded in each neuron and letting it modify its own synaptic weights based on its inputs and outputs (hebbian, modulated…).

Our brains do not have an external supervisor, our brain, the biological neurons that compose it, use different types of unsupervised learning, in a sense that they have plasticity and they change based on their experience. There is evidence that the hippocampus plays a very important role [13] in the formation of new memories, which means that a neural circuit like the hippocampus can modulate, or affect the topology and neural weights located in other parts of the brain, other neural networks. Thus, in a sense there is also modulation of learning algorithms at a more global scale of the neural network, and not just at the level of single neurons. Our brains of course have evolved the different features, the different rates of neural learning through experience, and the different neural circuits within our brain which affect and modulate other parts of our brain...

Though we could include a form of hebbian learning in neurons (discussed next), and create a large homogeneous hebbian or kohonen neural network... it will still be nothing more than a clustering network, there will be no self awareness within it. To create a truly intelligent neurocomputing system, we need to combine static neurons, neurons with plasticity, and different forms of unsupervised learning algorithms... all into a vast neural network. And combine it in a way that all these different parts work together perfectly, and allow for the whole emergent NN system to truly learn, which is the case with evolved biological neural networks.

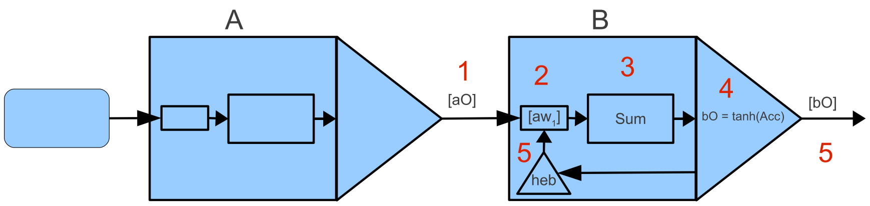
In this section we cover the unsupervised learning approaches, how to make neurons plastic, how to allow neurons to change their own weights through experience... The actual method of putting all these various systems together into a vast network that can have the potential of true learning, will be the subject of the rest of this book, with the method taken to accomplish this goal, being evolution. For the sake of exposure, and because we will use these particular unsupervised forms of NN learning once we've developed our basic neuroevolutionary platform and began expanding it beyond the current state of the art, we will briefly cover 4 particular unsupervised learning algorithms next.

2.6.1 Hebbian Learning

In 1949 Donald Hebb proposed a computational algorithm to explain memory and the computational adaptation process within the brain, he proposed a rule we now refer to as the Hebbian learning. The Hebbian learning is a neural learning algorithm that emulates plasticity exhibited by neurons, and which has been confirmed to a great extent to exist in the visual cortex [14].

As Hebb noted [26], "The general idea is an old one, that any two cells or systems of cells that are repeatedly active at the same time will tend to become 'associated', so that activity in one facilitates activity in the other.", or more concisely: "neurons that fire together, wire together".

The basic Hebbian rule for associative learning can be written as follows: For every weight w(i) in a neuron B, we add to the weight w(i) the value dw(i) where *dw(i) = x(i)*O*. This equation simply states that if we have a neuron B, which is connected from a number of other elements, and which produces an output *O* after processing the presynaptic *x(i)* input signals, and has a weight *w(i)* for every input signal x(i), then the change in the weight w(i) is x(i)*O. We can see that if both x(i) and O have the same sign, then the change in synaptic weight is positive and the weight will increase, whereas if x(i) and O are of opposite signs, then the weight will decrease for the synaptic connection between neuron B and the presynaptic element which sent it the signal x(i). So then for example, imagine that we have a NN with 2 neurons, in which neuron A is connected to neuron B. If neuron A sends a positive signal to neuron B, and this makes neuron B output a positive signal, then B's synaptic weight for the connection coming from A increases. On the other hand, if A's signal to B makes B produce a negative signal, then B's synaptic weight associated with A's signals is lowered. In this manner the two neurons synchronize. Fig-2.13 demonstrates this scenario and shows the hebbian rule in action.



1. Neuron A sends the vector signal [aO] to B.
2. Signal aO is weighted with B's synaptic weight aw$_1$.
3. The weighted signals (in this case just one) are summed together to produce the value: Acc.
4. Activation function is applied to Acc to produce B's output signal bO.
5. B outputs vector signal [bO], while at the same time uses the Hebbian rule to produce a delta w, and update the synaptic weight aw$_1$.

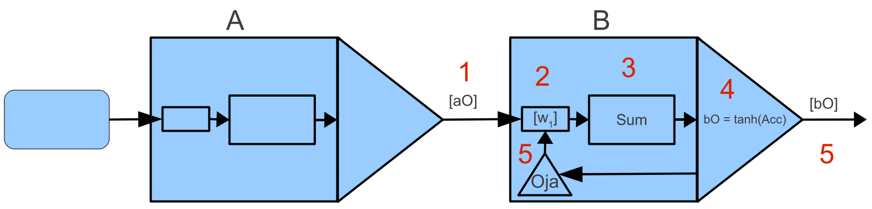
Update Rule: $U\_W_i = W_i + n*X_i*O$
Where X_i is the presynaptic signal associated with synaptic weight W_i, and where O is the postsynaptic neuron's output, and n the learning parameter.

Example: aw$_1$ = 0.5, aO = 1, n = 1
1. Neuron A sends the vector signal [1] to B.
2. Signal 1 is weighted with B's synaptic weight 0.5 to produce $Y_1 = x_1*aw_1 = 1*0.5 = 0.5$.
3. The weighted signals (in this case just one, Y_1) are summed together: Acc = Sum(Y_1) = 0.5.
4. Activation function tanh is applied to Acc to produce B's output signal bO = tanh(Acc) = 0.46.
5. B outputs the vector signal [bO] = [0.46], while at the same time uses the Hebbian rule to produce: dw = 0.46 = 0.46*1, and update the synaptic weight aw$_1$. Thus, the updated aw$_1$ = 0.5 + 0.46 = 0.96. The new synaptic weight is: **aw$_1$ = 0.96.**
If we now continue running this update rule, with A firing signals of the same magnitude, 1, the sequence of B's weight aw$_1$ is: **0,5, 0.962, 1.71, 2.64, 3.63, 4.63**
The synaptic weight continues to increase in magnitude over time.

Fig. 2.13 Neuroplasticity through Hebbian learning.

In the above figure, we can see that just from one signal coming from Neuron *A*, a signal that was positive and thus producing a positive delta weight with regards to the positive synaptic weight of B, B's neural weight nearly doubled for the connection with *A*. A few more signals from *A*, and the weight aw1 would have grown significantly larger, and eventually drowned out any other weights to other links. Thus the problem with the original and very simple Hebbian learning rule is that it is computationally unstable. For example, as noted, the weights do not saturate, they can continue growing indefinitely. If that does occur, then the weights that grow fastest will eventually drown out all other signals, and the output of the neuron, being a sigmoid of tanh, will always be 1. Thus eventually the neuron will stop truly discerning between signals, since its weights will be so large that no matter the input, 1 will always be the neuron's output. Another problem is that, unlike in a biological neuron, there is no weight decay in the original Hebb's rule, there is no way for the synaptic weights for the incoming signals to become weaker.

New learning algorithms that fix computational instabilities of the original Hebbian rule have been created. For example, three versions of such rules are the Oja's rule [15], the Generalized Hebbian Algorithm (GHA) aka Sanger's rule [16], and the BCM rule [17]. The Oja's and BCM rules in particular, incorporate weight decay, and are more biologically faithful. In Fig. 2.14 I demonstrate how a neuron using the Oja's learning algorithm updates its synaptic weights after having processed its input vector.



1. Neuron A sends the vector signal [aO] to B.
2. Signal aO is weighted with B's synaptic weight aw_1.
3. The weighted signals (in this case just one) are accumulated.
4. Activation function is applied to produce B's output signal bO.
5. B outputs vector signal [bO], while at the same time uses the Hebbian rule to produce a delta w (dw_i), and update the synaptic weight w_i.

Update Rule: Updated_W_i = W_i + n*O*(X_i-O*W_i)
Where X_i is the presynaptic signal associated with synaptic weight W_i, where O is the postsynaptic neuron's output, and n the learning parameter.

Example: aw1 = 0.5, aO = 1, n = 1
1. Neuron A sends the vector signal [1] to B.
2. Signal 1 is weighted with B's synaptic weight 0.5 to produce $Y_1 = w_1 * X_1 = 1*0.5 = 0.5$.
3. The weighted signals (in this case just one) are summed together to produce: Acc = Sum(Y_1) = 0.5
4. Activation function tanh is applied to Acc to produce B's output signal bO = tanh(Y_1) = 0.46.
5. B outputs the vector signal bO = [0.46], while at the same time uses Oja's rule to produce a dw_1 = 1*0.46*(1-0.46*0.5) = 0.35 , and update the synaptic weight w_1.
Thus updated $w_1 = w_1 + dw_1$ = 0.5 + 0.35 = **0.85.** If we now continue running this circuit few dozen times with A always producing the same signal, the sequence of B's weight w_1 is: **0.5, 0.856, 1.14, 1.2, 1.2...**
The synaptic weight naturally becomes saturated

Fig. 2.14 Neuroplasticity through Oja's rule.

As can be seen in Fig-2.14, unlike the original Hebbian rule, Oja's rule produces a smaller weight increase, but more importantly, if we process a few more signals, then we would notice that the weight does not grow indefinitely. This computational system is stable, the weight eventually saturates at some viable value, and if that particular synapse, and thus the synaptic weight associated with it, is not stimulated any further by incoming signals (the incoming signals are not as high in magnitude), the weight begins to decay, memory slowly deteriorates.

The only problem is that, if there were to have been more than one synaptic weights (w1, w2...wi), they would all still follow the same type of rule, the same learning rate 'n'. To make that rule even more flexible, we can employ neuromodu–lation, which allows for every synaptic weight to update differently from every other, making the plasticity of the neuron even more flexible and realistic. This form of unsupervised learning is discussed next.

2.6.2 Neuromodulation

In biological neural networks, neuromodulation refers to the process of the release of several classes of neurotransmitters into the cerebrospinal fluid, which then modulate a varied class of neurons within reach of the released neurotransmitters. In this manner, a neural circuit that releases the neurotransmitters into the cerebrospinal fluid, can affect some area of neural tissue, augmenting its behavior by making it more easily exited or inhibited for example. Neuromodulation can also be direct, when one neuron is connected to another, and depending on this modulatory neuron's signals, the behavior of the modulated neuron, the way it processes information, is modified.

In artificial neural networks, the same can be accomplished. We can allow a neuron or a neural circuit to use its output to modulate, or control the plasticity type and the adaptation pace (learning parameter for example) of another neuron or neural circuit. Thus for example assume that we have 2 neural circuits, A and B, which form a neural network based system. Circuit A is connected from a set of sensors, and to a set of actuators. Circuit B is also connected to the same set of sensors, but its output signals, instead of going to the actuators, are used to modulate and dictate how the weights of the neurons in circuit A change and adapt over time, as shown in Fig-2.15. Thus, in this neural network system circuit B modulates circuit A, and controls that circuit's ability to learn, pace of learning, and the learning algorithm in general. Since a neural network is a universal function approximator, this type of learning algorithm can be highly versatile and robust, and the modulatory signals produced by the neural circuits can be of any form.

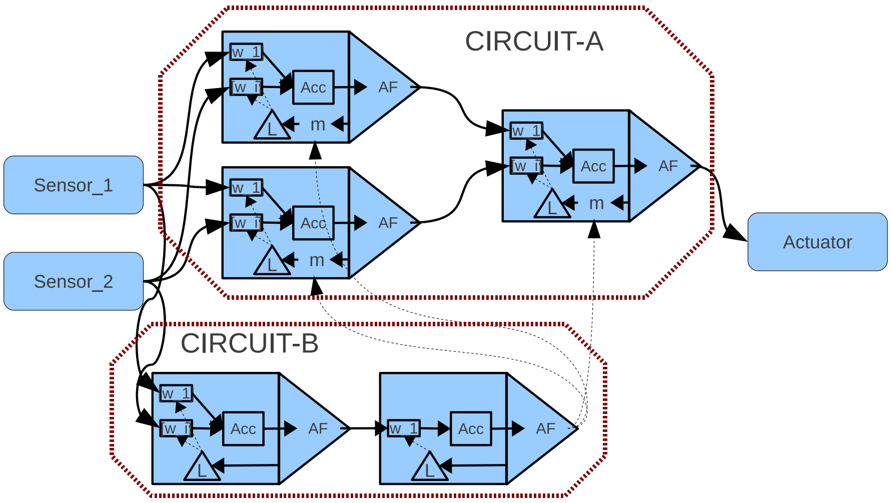


Fig. 2.15 A Neural Network based system with plasticity through neuromodulation. In this figure, Circuit-B modulates Circuit-A's learning algorithm.

The equation used to add plasticity to a neuron through neuromodulation is: *DWij* = ***L*** = *f*N(A*Oi*Oj + B*Oi + C*Oj)*. DWij is the delta weight, change in the synaptic weight of neuron j for the link coming from neuron i. N is the learning rate, which dictates the general magnitude of weight change after the neuron processes a signal. A, B, and C are parameters weighting the contribution of the output signal coming from the presynaptic element i and the output signal produced by the postsynaptic neuron j, and together forming the non linear plasticity factor. Finally, the value **f** is a further modulatory signal which dictates how rapidly, and in what direction the weight will change based on the learning rule **L.** In standard neuromodulation, the value f is produced by the modulating neuron or neural circuit, and the parameters N, A, B, and C are set by the researcher, or evolved and optimized through a neuroevolutionary process. But the parameters N, A, B, and C can also be produced by the modulatory neural circuit B in vector form for each neuron in circuit A, to modulate and give those neurons even more dynamic neuroplasticity.

In a sense, we can think of Circuit-B as being the biological part of circuit A, that part which produces plasticity. We can recreate the neural network shown in Fig-2.15 to be composed not of two separate neural circuits, one which does the actual processing (A) and one which does neuromodulation (B), but instead composed of one neural network, where every neuron has an embedded circuit B, which gives it plasticity. This type of neural architecture is shown in Fig-2.16.

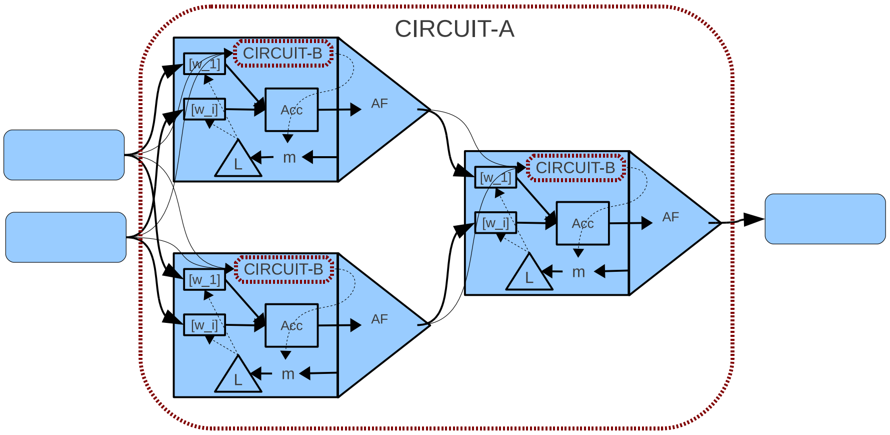


Fig. 2.16 Another type of neuromodulatory architecture.

From the above figure, we can see that each neuron now has the functionality of Circuit-B embedded inside of it. Also, there is a small change in how this new Circuit-A functions. The embedded Circuit-B does not use as input the signals coming from the two sensors, but instead uses as its input the same input as the neuron to which it adds plasticity. In this manner the modulatory circuit sees the input signals of the neuron which it modulates, making the modulation signal specific to the data of the neuron in which it is embedded.

An actual example of the steps taken in processing signals by a neural network with plasticity shown in Fig-2.15 is presented in Fig-2.17. The sequence of events in such a NN is demonstrated by the numbers given for the various steps, and is further elaborated in the following list:

1. The two sensors produce signals, and forward them to the neurons in the first layers of Circuit-A and Circuit-B.
2. The two neurons of Circuit-A process the signals from the two sensors, and produce outputs. The neuron of Circuit-B also at the same time processes the signals from the two sensors, producing the output and forwarding it to the neuron in the next neural layer of Circuit-B.
3. The second neuron in the Circuit-B processes the signal coming from the presynaptic neuron.
4. Circuit-B produces the modulatory signal, sending it to all neurons of Circuit-A. Since the first two neurons in Circuit-A have already processed their input signals, they use this modulatory signal and then do both, update their synaptic weights based on this modulatory signal, and update their learning rule parameters, where the used learning rule might be: General Hebbian, Oja's Rule, or some other.
5. The neuron in the second layer of Circuit-A produces an output after processing the signals sent to it by the two presynaptic neurons in the first layer of Circuit-A.
6. The neuron in the second layer of Circuit-A uses the modulatory signal sent to it by Circuit-B in step-4 to update its synaptic weights, and learning rule parameters.
7. The sensors produce another set of signals and forward those signals to the neurons they are connected to. The loop repeats itself.

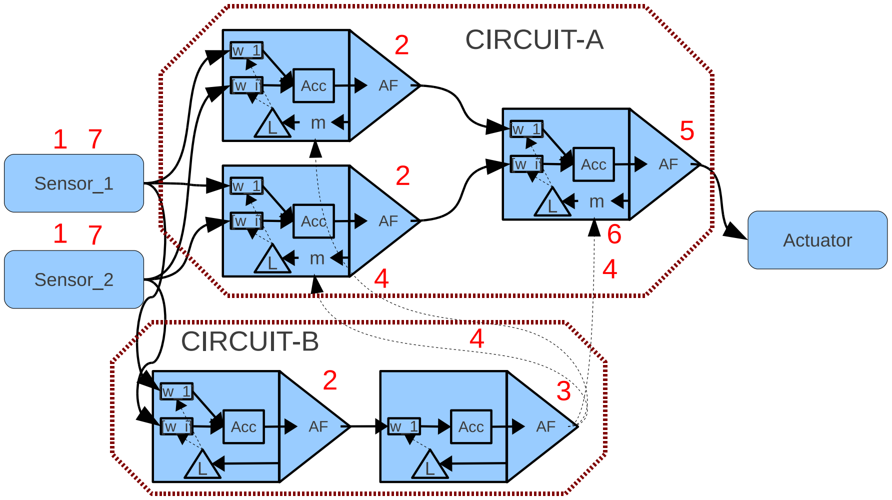


Fig. 2.17 Neuromodulation in action.

At this point you are probably asking yourself the following question: *Sure, now we can allow for some neuron to learn and adapt, to possess plasticity... but plasticity is controlled by another type of neural network, so how do we set up that other neural network's synaptic weights and parameters so that it can actually produce the modulatory signals that are useful in the first place?* That is a valid question, in fact, for example in the figure above where we modulate two neurons, instead of having to set up those neuron's synaptic weights, we have to set up the weights of the neurons in Circuit-B, each possessing 2 weights. We can do this through evolution. Evolution can optimize the synaptic weights and the various parameters needed by the modulatory neural circuits, which would then modulate effectively the other neural circuits of the complete neural network.

Unlike the static simple neurons, the neurons shown in the above figure are complex, plastic, but highly robust and adaptive elements. A neural network of such elements, evolved to work coherently as biological neural networks do, would have quite a significant amount of learning ability. We will build such systems and variants of it in later chapters, we will embed such adaptive and plastic neural networks in artificial organisms when we'll apply our neuroevolutionary system to ALife simulations, and as you will see, such systems do indeed have high potency, and might be exactly the building blocks needed when the goal is to evolve an intelligent neurocognitive system.

2.6.3 Competitive Learning

Competitive Learning [18] is another form of unsupervised learning, but unlike the Hebbian and the neuromodulation methods which add plasticity to each neuron, this one requires some system/process that has a global view of all the neurons forming the neural network undergoing competitive learning. In competitive learning we have a set of neurons, each of which is connected to a given list of sensors, and where each neuron competes with the others for the right to respond to a subset of sensory signals. Over time, competitive learning increases the specialization of each neuron for some particular set of signals, and thus allows the NN to act and spontaneously form a clustering/classification network.

A NN which uses competitive learning (CL) is realized through the implementation of the following set of steps:

1. Choose *j* number of sensors whose signals you wish to cluster or classify.
2. Create *i* number of neurons, each connected to all j sensors, and each neuron using a random set of synaptic weights.
3. **DO:**
    1. Propagate the signals from sensors to the neurons.
    2. Each neuron processes the sensory signals and produces an output signal.
    3. An external CL process chooses the neuron with the highest output signal magnitude.

4. The CL updates the synaptic weights of that neuron by applying to it a form of Hebbian learning (by using the Oja's rule for example).

4. **UNTIL:** The network begins to cluster signals, and a pattern begins to emerge.

This is a simple learning rule that can be used to see if there is a pattern in the data, and if those signals can be clustered. Fig-2.18 shows a diagram of a NN system utilizing the competitive learning algorithm.

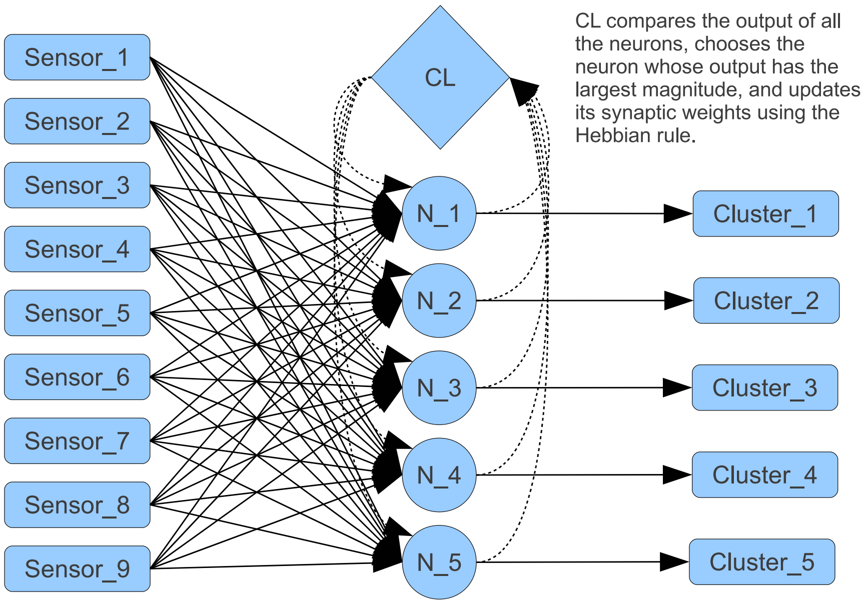

Sensor_1
Sensor_2
Sensor_3
Sensor_4
Sensor_5
Sensor_6
Sensor_7
Sensor_8
Sensor_9
CL
CL compares the output of all the neurons, chooses the neuron whose output has the largest magnitude, and updates its synaptic weights using the Hebbian rule.
N_1
N_2
N_3
N_4
N_5
Cluster_1
Cluster_2
Cluster_3
Cluster_4
Cluster_5

Fig. 2.18 A neural network employing competitive learning.

2.6.4 Kohonen/Self Organizing Map

A Kohonen map [19], also known as a self organizing map (SOM), is a type of neural network that in a sense represents a hypercube or a multidimensional grid of local functions, and through the use of a form of competitive learning the SOM performs a mapping of data from a high dimensional space into a lower dimensional one, while preserving that data's topology. These types of neural networks originated in the 80s and are loosely based on associative memory and adaptive learning models of the brain. Like the competitive learning neural network, a SOM system requires a process that has a global view of the NN, so that learning can be achieved. An example of a 2d SOM system is shown in Fig-2.19.

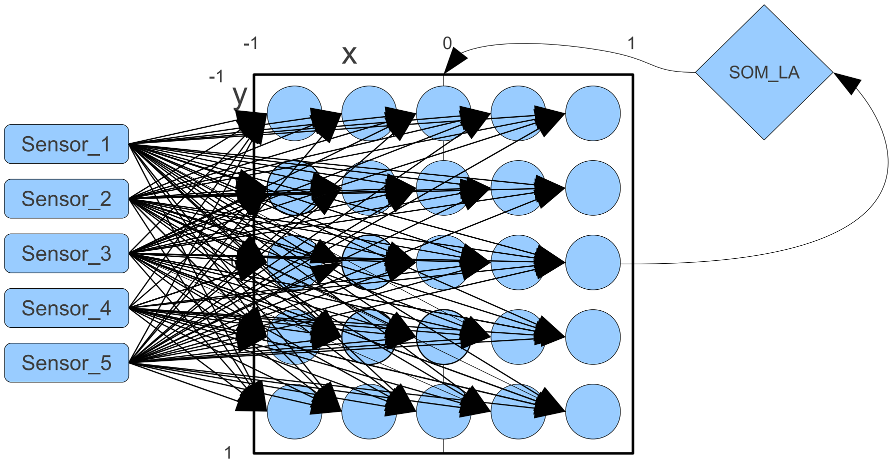


Fig. 2.19 A self organizing map, where the SOM_LA process performs SOM based Learning Algorithm computations, and synaptic weight updates.

To set up a Kohonen map we create a hypercube based substrate with embedded neurons within, where each neuron has a set of weights and a coordinate within the substrate. Each axis of the hypercube ranges from -1 to 1, and the neurons are embedded regularly within the substrate (this is somewhat similar to the hypercube representation we discussed in Chapter 1.2.10, which is used by the HyperNEAT system). The actual density, the number of neurons forming the SOM, is set by the researcher. Finally, each neuron in this hypercube is connected to the same list of sensors.

The learning algorithm used by a SOM is somewhat similar to one utilized by the competitive learning we discussed in the previous section. When the sensors propagate their vectors to the neurons, we check which of the neurons within the hypercube has a weight vector which is closest to the input vector based on a Cartesian distance to it. The neuron whose weight vector is the closest to the input vector is called the best matching unit, or BMU. Once this neuron is found we apply the weight update rule: $\boldsymbol{Wv}(t + 1) = \boldsymbol{Wv}(t) + \Theta(d)*\alpha(t)*(\boldsymbol{I}(t) - \boldsymbol{Wv}(t))$, to all neurons in the hypercube, where **Wv**(t+1) is the updated weight vector, **Wv**(t) is the neuron's weight vector before the update, α(t) is a monotonically decreasing learning coefficient similar to the one used in simulated annealing [20,12], **I**(t) is the input vector, and Θ(d) is usually the Gaussian or the Mexican-Hat function of the distance between the BMU and the neuron in question (thus it is greatest for the BMU neuron, and decreases the further you move away from the BMU). Once the new weight vector is calculated for every neuron in the hypercube, the sensors once again fanout their sensory vectors to the neurons. We continue with this process for some maximum X number of iterations, or until α(t) reaches a low enough value. Once this occurs, the SOM's output can be used for data mapping. A trip through a single iteration of the SOM learning algorithm is shown in Fig-2.20.

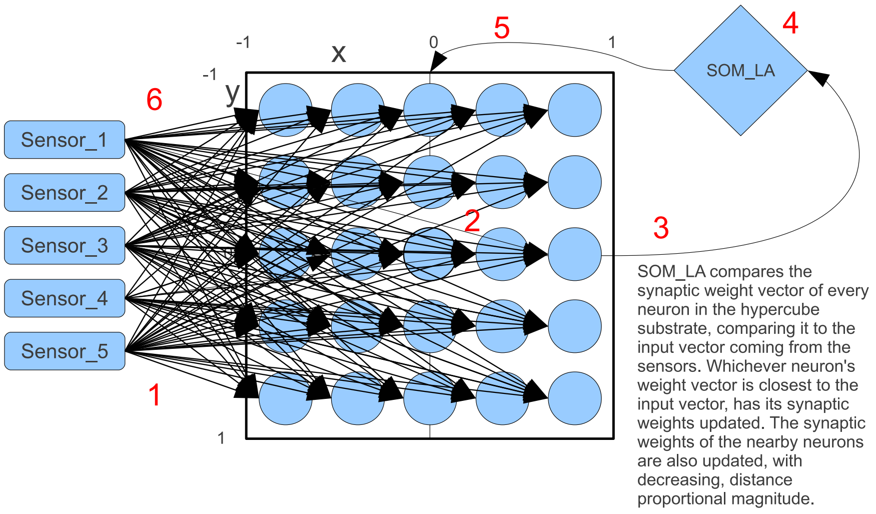

-1
x
0
5
1
4
SOM_LA
-1
y
6
Sensor_1
Sensor_2
Sensor_3
Sensor_4
Sensor_5
2
3
1
1
SOM_LA compares the synaptic weight vector of every neuron in the hypercube substrate, comparing it to the input vector coming from the sensors. Whichever neuron's weight vector is closest to the input vector, has its synaptic weights updated. The synaptic weights of the nearby neurons are also updated, with decreasing, distance proportional magnitude.

Fig. 2.20 Self Organizing Map in action.

The following list elaborates on each of the algorithm steps in the above figure:

1. The sensors forward their signals to all the neurons in the substrate (each neuron has its own coordinate).
2. Each neuron in the substrate processes the incoming signals, and produces an output.
3. A process by the name SOM_LA which has a global view of all the neurons in the substrate, compares the neural weights to the input vectors for every neuron.
4. SOM_LA finds the neuron whose synaptic weight vector is closest to the input vector coming from the sensors.
5. SOM_LA updates that neuron's synaptic weights, and updates the synaptic weights of the neurons around it, with the synaptic weight update decreasing in magnitude proportionally to the distance of those other neurons to the winning/chosen neuron.
6. The sensors forward their signals to all the neurons in the substrate... The loop repeats itself.

There are numerous variations on the original Kohonen map, for example the General Topographic Map (GTM) and the Growing Self Organizing Map (GSOM), are two of such advanced self organizing maps.

2.6.5 Putting it All Together

In this chapter we have discussed 4 different types of unsupervised learning algorithms. There are of course many others, like the Hopfield memory network that models associative memory, and the Attenuated Resonance Theory (ART) NN that models a scalable memory system. We can see that such unsupervised learning algorithms add plasticity to the neurons, and the neural networks in general. But, these types of learning algorithms themselves have parameters that need to be set up before the system can function. And what about the general topology of the NNs which possess plasticity? After all, we can't simply add some unsupervised learning algorithm to a random NN structure, and then expect it to immediately possess intelligence. The way neurons are connected to one another in the NN, the topology itself, is just as important, if not more so, than the synaptic weights of the neurons. A neurocognitive system possessing intelligence will certainly have to utilize many of these types of NNs and the different plasticity types they possess. This possible future neurocognitive system will integrate all these learning neural circuits into a single, cohesive, synchronized, vast neural network system, possessing the topology and architecture similar to an example shown in Fig-2.21.

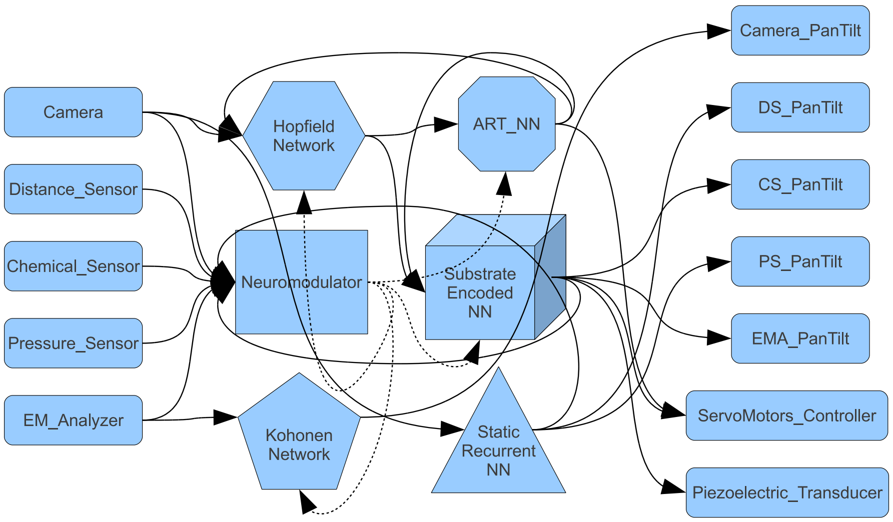


Fig. 2.21 A possible vast NN composed of neurons with and without plasticity, and different interconnected neural circuit modules. The flexibility of an evolved NN based system which draws upon and uses all the available learning algorithms, encodings, neuron types... could potentially be immense.

How can we figure out how to put these modules together, how to connect the neurons in the right manner, how to bootstrap a NN system so that it can take over from there, and so that its own intelligence and ability to learn can continue the work from that point onwards? That problem has already been solved once before, we are the result; the solution is evolution.

2.7 Summary

In this section we have discussed how biological neurons process information, their ability to integrate spatiotemporal input signals, and change their signal processing strategy, a process called *plasticity*. We then discussed how artificial neural networks process signals, and that the most common such neural networks deal with amplitude encoded signals, rather than frequency encoded signals as is the case with biological neural networks. Although as noted, there are artificial neural networks called spiking neural networks, which like biological NNs deal with frequency encoded signals.

We then discussed the various topologies, architectures and NN plasticity rules. We discussed how a recurrent NN exhibits memory, and how the Hebbian, Oja's, and neuromodulation learning rules allow for NNs to adapt and change as they interact and process signals. Finally, we discussed how the various parameters and topologies of these NNs can be set, through evolution, allowing for the eventual vast NN to incorporate all the different types of learning rules, plasticity types, topologies, and architectures.

With this covered, we move to the next chapter which will introduce the subject of evolutionary computation.

2.8 References

[1] The Blue Brain Project: http://bluebrain.epfl.ch/
[2] Dawkins R (1976) The Selfish Gene. (Oxford University Press), ISBN 0192860925.
[3] Dawkins R (1982) The Extended Phenotype. (Oxford University Press), ISBN 0192880519.
[4] Hornik K, Stinchcombe M, White H (1989) Multilayer Feedforward Networks are Universal Approximators. Neural Networks 2, 359-366.
[5] Sher GI (2010) DXNN Platform: The Shedding of Biological Inefficiencies. Neuron, 1-36. Available at: http://arxiv.org/abs/1011.6022.
[6] Parisi D, Cecconi F, Nolfi S (1990) Econets: Neural Networks That Learn in an Environment. Network Computation in Neural Systems 1, 149-168.
[7] Predators and Prey in simulated 2d environment, Flatland: http://www.youtube.com/watch?v=HzsDZt8EO70&list=UUdBTNtB1C3Jt90X1I26Vmhg&index=2&feature=plcp
[8] Hassoun MH (1995) Fundamentals of Artificial Neural Networks. (The MIT Press).
[9] Lynch M (2007) The Origins of Genome Architecture S. Associates, ed. (Sinauer Associates Inc).
[10] Haykin S (1999) Neural Networks: A Comprehensive Foundation J. Griffin, ed. (Prentice Hall).
[11] Rojas R (1996) Neural Networks: A Systematic Introduction. (Springer).
[12] Gupta MM, Jin L, Homma N (2003) Static and Dynamic Neural Networks From Fundamentals to Advanced Theory. (John Wiley & Sons).
[13] Di GG, Grammaldo LG, Quarato PP, Esposito V, Mascia A, Sparano A, Meldolesi GN, Picardi A (2006) Severe Amnesia Following Bilateral Medial Temporal Lobe Damage Occurring On Two Distinct Occasions. Neurological sciences official journal of the Italian Neurological Society and of the Italian Society of Clinical Neurophysiology 27, 129-133.

[14] Kirkwood A, Rioult MG, Bear MF (1996) Experience-Dependent Modification of Synaptic Plasticity in Visual Cortex. Nature 381, 526-528.

[15] Oja E (1982) A Simplified Neuron as a Principal Component Analyzer. Journal of Mathematical Biology 15, 267-273.

[16] Sanger T (1989) Optimal Unsupervised Learning in a Single-Layer Linear Feedforward Neural Network. Neural Networks 2, 459-473.

[17] Bienenstock EL, Cooper LN, Munro PW (1982) Theory For The Development of Neuron Selectivity: Orientation Specificity and Binocular Interaction in Visual Cortex. Journal of Neuroscience 2, 32-48.

[18] Rumelhart DE, McClelland JL (1986) Parallel Distributed Processing M.I.T. Press, ed. (MIT Press).

[19] Kohonen T (1982) Self-Organized Formation of Topologically Correct Feature Maps. Biological Cybernetics 43, 59-69.

[20] Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by Simulated Annealing. Science 220, 671-680.

[21] Cerny V (1985) Thermodynamical Approach to The Traveling Salesman Problem: An Efficient Simulation Algorithm. Journal of Optimization Theory and Applications 45, 41-51.

[22] An excellent discussion of neuron and synapse: http://en.wikipedia.org/wiki/Neuron

[23] Gerstner W (1998) Spiking Neurons. In Pulsed Neural Networks, W. Maass and C. M. Bishop, eds. (MIT-Press), pp. 3-53.

[24] Ang CH, Jin C, Leong PHW, Schaik AV (2011) Spiking Neural Network-Based Auto-Associative Memory Using FPGA Interconnect Delays. 2011 International Conference on FieldProgrammable Technology, 1-4.

[25] Qingxiang W, T MM, Liam PM, Rongtai C, Meigui C (2011) Simulation of Visual Attention Using Hierarchical Spiking Neural Networks. ICIC: 26-31

[26] Hebb DO (1949) The organization of behavior. Wiley, eds. (Wiley)