

Connectionist Models: The Briefest Course

Robert M. French
Cognitive Science, Dept. of Psychology,
University of Liège, Belgium
email: rfrench@ulg.ac.be
URL: <http://www.ulg.ac.be/cogsci/rfrench.html>

“Systems that are deliberately constructed to make use of some of the organizational principles that are felt to be used in the human brain.” (Anderson & Rosenfeld, 1990, *Neurocomputing*, p. xiii)

Brief Historical Overview of the Origin of Connectionist Networks

McCulloch & Pitts (1943, 1947):

The “essential” neuron as a logic gate, collections of neurons, appropriately wired together, can do logical calculus. This can be extended to systems of neurons (e.g., thalamo-cortical relays) in the real brain.

Hebb (1949): *The Organization of Behavior*.

- learning rule of synaptic reinforcement. The axon of neuron A is separated by synapses from the dendrites of neuron B. When neuron A fires and this is followed immediately by the firing of neuron B, the synapse between the two neurons is strengthened, i.e., the next time A fires, it will be easier for B to fire. In Hebb’s model there are only excitatory synapses.

- persistence of activity to form cell assemblies corresponding to concepts. Note that these cell assemblies can overlap. e.g., the cell assembly associated with “dog” will have considerable overlap with those associated with “wolf”, “cat”, etc.

- recruitment (gathering of cells to form cell assemblies – creation of a cell assembly corresponding to a new concept) and fractionation (creation of new cell assemblies from an old one, corresponding to the refinement of a concept).

Rochester, Holland, Haibt, & Duda (1956).

Even though Minsky (1951) built the first reinforcement learning machine with 40 interconnected units, the first real simulation that attempted to implement the principles outlined by Hebb in real computer hardware was done by Rochester, Holland, Haibt, & Duda (1956). The authors attempted to simulate the emergence of cell assemblies in a small network of 69 neurons. They found that “the detailed patterns of firing diverged rapidly.” In other words, everything became active in their network. Upon conferring with Hebb and his coworker, Milner, they decided that the network needed to include inhibitory synapses as well. This worked and cell assemblies did, indeed, form. This is probably the earliest example in neural network modeling of a network which made a prediction (i.e., inhibitory synapses are needed to form cell assemblies), that was later confirmed in real brain circuitry.

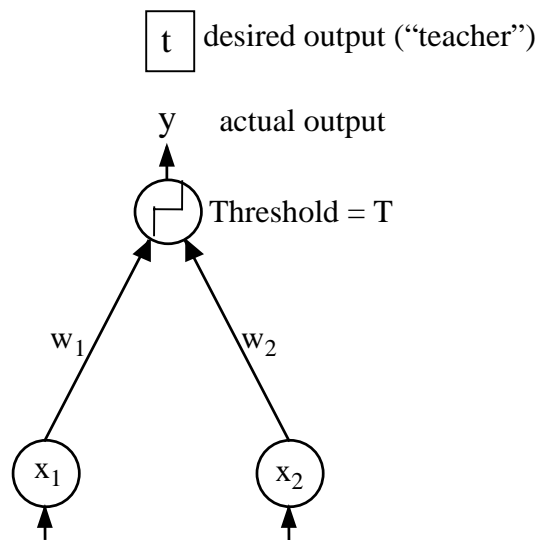
Rosenblatt (1958, 1962): The Perceptron

Rosenblatt's perceptron created a sensation when it appeared. It could learn to associate inputs with outputs. He believed that this was how the visual system could learn to associate low-level visual input with higher level concepts. He introduced a learning rule (weight-change algorithm) that allowed the perceptron to learn these associations. He showed (Perceptron convergence theorem) that if it was possible for a perceptron to learn (i.e., there existed some set of weights that allowed the perceptron to correctly associate a given input with a particular output), then the perceptron would, in fact, always learn the association.

The most elementary perceptron

There are:

- two layers of nodes (one layer of weights)
- only feedforward connections
- a threshold function on each output unit
- a linear summation of the weights times inputs.



$$\text{if } \sum_{i=1}^{\text{INPUTS}} w_i x_i > \text{Threshold} \text{ then } y = 1$$
$$\text{else } y = 0$$

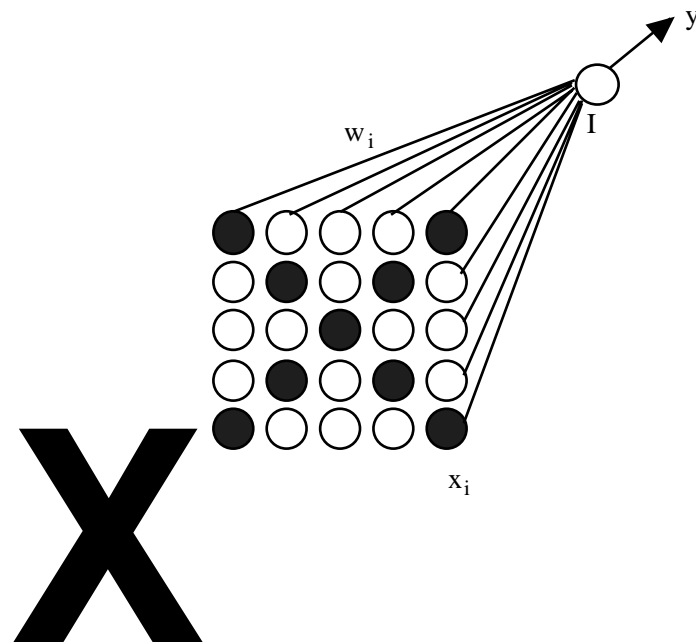
The perceptron learning rule (weight-change rule) is:

$$w_{\text{new}} = w_{\text{old}} + \beta x(t - y)$$

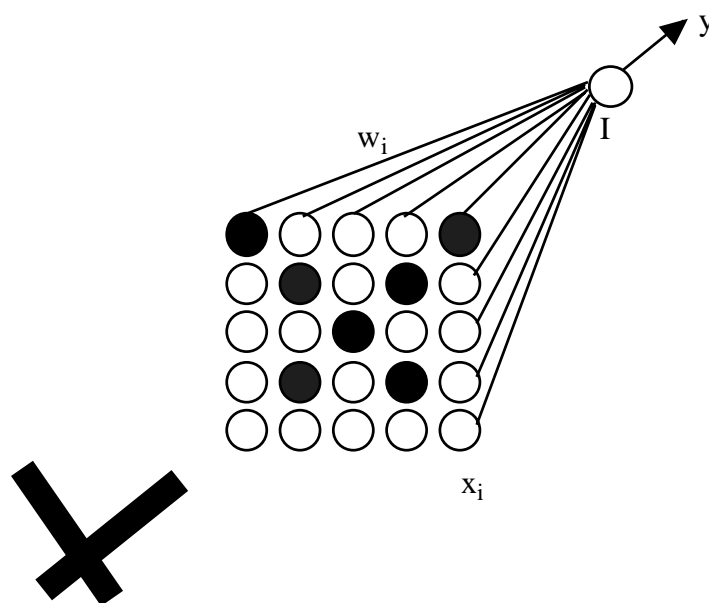
where

β is the learning constant, $0 < \beta < 1$

There are, of course, many other possible learning rules for changing weights but this was the original weight-change rule used by Rosenblatt.



This perceptron learns to associate the visual input of two crossed straight lines with the character "X". In other words, the output y of the network will be the character "X".

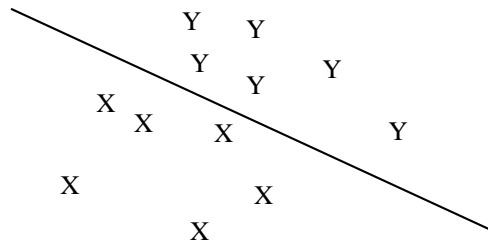


Here we can see that that real image in the world is degraded, but if the network has already learned to correctly identify the original complete “X”, it will recognize the degraded X as being an “X”.

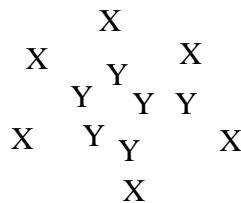
Fundamental limitations of the perceptron

With the arrival of the 1960’s came a new paradigm, “symbolic” AI. The two paradigms, the perceptron and symbolic AI co-existed until 1969 with the appearance of *Perceptrons* by Minsky and Papert.

They showed that the Rosenblatt two-layer perceptron could only classify linearly separable sets. In other words, it could



In this case, it could correctly associate the “X”’s with the one category and the Y’s with a second category.



However, in this case, no perceptron could separate the X’s and Y’s into two separate categories because the set of X’s is not linearly separable from the set of Y’s.

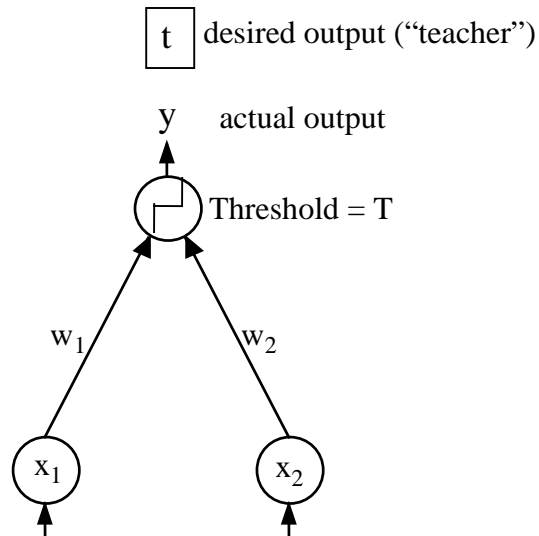
The (infamous) XOR problem

Minsky and Papert claimed that there were a number of extremely simple types of associative patterns that the perceptron, or any series of perceptrons, would not be able to learn. The simplest example was the XOR Problem. The simple perceptron with 2 inputs, one output and two weights could never learn the XOR associations. Since XOR was, according to Minsky & Papert, an elementary logical operation on which many more were built, this severely weakened the perceptron’s claim to be able to do general cognition. Further, they also showed the parity detection (“Are there an odd or an even number of dots?”), contiguity detection (Is this shape contiguous or not?), etc. could not be done by the perceptron.

First, here is the function that must be learned by the perceptron (XOR).

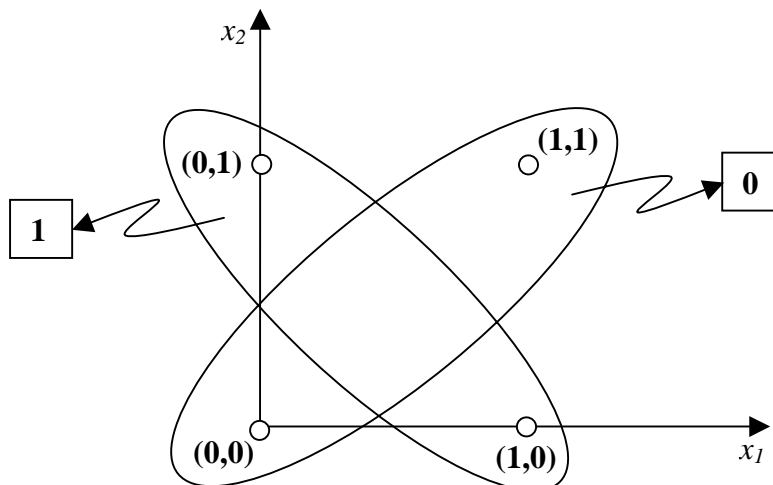
Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

The problem is to find a set of weights w_1 and w_2 and a threshold T , such that the perceptron below can learn the above XOR function.



The proof that there cannot be such a set of weights and a threshold that satisfy the condition is as follows.

The activation arriving at the output node is defined by $w_1x_1 + w_2x_2$. If this value exceeds T , then the perceptron outputs 1, otherwise 0. In other words, if $w_1x_1 + w_2x_2 > T$ then we output 1, otherwise 0. But $w_1x_1 + w_2x_2 = T$ is a straight line if we consider x_1 and x_2 to be the axis of a coordinate system.



There are clearly no values of w_1 , w_2 and T such that a straight line of the form $w_1x_1 + w_2x_2 = T$ can cut this graph so that (0,0) and (1,1) will be on one side of the line (i.e., the region for which the network will output 0) and (0,1) and (1,0) will be on the other side of the line (i.e., the region for which the network will output 1).

Thus came to a screeching halt virtually all of the research on perceptrons for almost a decade and a half.

The Revival of the (Multi-layered) Perceptron: The Connectionist Revolution (1985) and the Statistical Nature of Cognition

Over the course of the next 15 years, in particular throughout the 1970's it became increasingly clear that the route to machine intelligence via traditional symbolic AI was going to be a lot harder than it had originally been thought. Minsky wrote in 1967, shortly before the publication of *Perceptrons*, "Within a generation the problem of creating 'artificial intelligence' will be substantially solved." A decade and a half later, in 1982, he was talking a different language, "The AI problem is one of the hardest ever undertaken by science."

"Simple" tasks that we humans do (almost) effortlessly, like face, word and speech recognition, like retrieving information in the presence of incomplete cues, generalizing, etc., proved to be notoriously hard for symbolic AI. Time-scale issues became important: Human cognition examined at level of seconds or minutes seemed highly sequential. But at the low-level, it was a massively parallel joint endeavor of innumerable processes.

Without explicitly denying the importance of symbolic models of certain cognitive activities (debugging a computer program, fixing a car, solving certain types of problems, etc. are profoundly symbolic, logic-based activities), the statistical nature of much of cognition became ever more apparent.

Furthermore, the symbols referred to in macrostructural (i.e., symbolic) models of cognitive processing were considered to be approximate descriptions of emergent properties of the microstructure. In other words, symbols *emerged* from low-level neural mechanisms.

At least three factors contributed to the revival of perceptron-based views of cognition:

- the radical failure of AI to achieve the goals announced in the 1960's
- the growing awareness of the statistical and "fuzzy" nature of cognition
- the development of improved perceptrons, capable of overcoming the problems brought to light by Minsky and Papert.

Neurally based models had the advantage of potentially being able to model not only "emergent" phenomena, but also "emerged" (i.e., symbolic) phenomena.

Advantages of connectionist models

Connectionist models had numerous advantages over their symbolic predecessors, among them:

- They were specifically designed *to learn*.
- They could achieve pattern completion of familiar patterns.
- They could generalize to novel patterns, based on previously learned patterns.

(Note: there is no specific “generalization” mechanism in these systems. The ability to generalize arises from the underlying architecture.)

- They were content addressable, in other words, it was possible to access information in memory based on nearly any attribute of the representation that the network was attempting to retrieve.

- They processed information in a massively parallel manner: Feldman & Ballard (1982) argue that neural hardware is too slow and too unreliable for sequential models of processing. We can do very complex processing in a few hundred ms. But transmission across a synapse ($\sim 10^{-6}$ in.) occurs in about ~ 1 ms. Thus, the tasks must be accomplished in no more than a few hundred serial steps, which is impossible.

- Graceful degradation: when they are damaged, their performance degrades gradually.

- Fault tolerance: they are highly redundant systems and thus fault tolerant.

If we look at real brains, we see a number of characteristics that serve as the basis of neural network design. Some of these are:

- Massive parallelism: neurons are slow but there are lots of them
- Neurons receive input from lots of other neurons.
- Learning involves modifying the strength of synaptic connections.
- Neurons communicate with one another via activation or inhibition.
- Connections in the brain have a clear geometric and topological structure.
- Information is continuously available to the brain.
- Graceful degradation of performance in the face of damage and information overload
- Control is distributed, not central (i.e., no central executive).
- One primary way of understanding what the brain does is relaxation to attractors.

General principles of all connectionist networks

There are seven major components of a connectionist system:

- a set of processing units
- a state of activation defined over all of the units
- an output function (“squashing function”) for each unit: Transforms unit activation into outgoing activation;
- a connectivity pattern with two features:
 - weights of the connections
 - locations of the connections

- an activation rule for combining inputs impinging on a unit to produce a total activation for the unit
- a learning rule, by which the connectivity pattern is changed.
- an environment in which the system operates (i.e., how is the i/o represented and given to/taken from the system)

Knowledge storage: Knowledge is stored exclusively in the *pattern of strengths of the connections (weights) between units*.

Learning: The system learns by *automatically adjusting the strengths of these weights* as it receives information from its environment. There are no high-level rules programmed into the system. **The network learns multiple patterns in the SAME set of connections**, which is precisely what makes generalization, graceful degradation, etc. relatively easy in connectionist networks; It is also what makes planning, logic, etc. are so hard.

Two classes of networks

The two major classes of connectionist networks are defined by the learning algorithms they use, namely:

- Supervised learning:

This class includes all error-driven learning algorithms. For these networks, there is an error signal associated with the desired output of each pattern to be learned. This error is a function of the difference between the desired output and the actual output of the network. This error is gradually decreased by the learning algorithm.

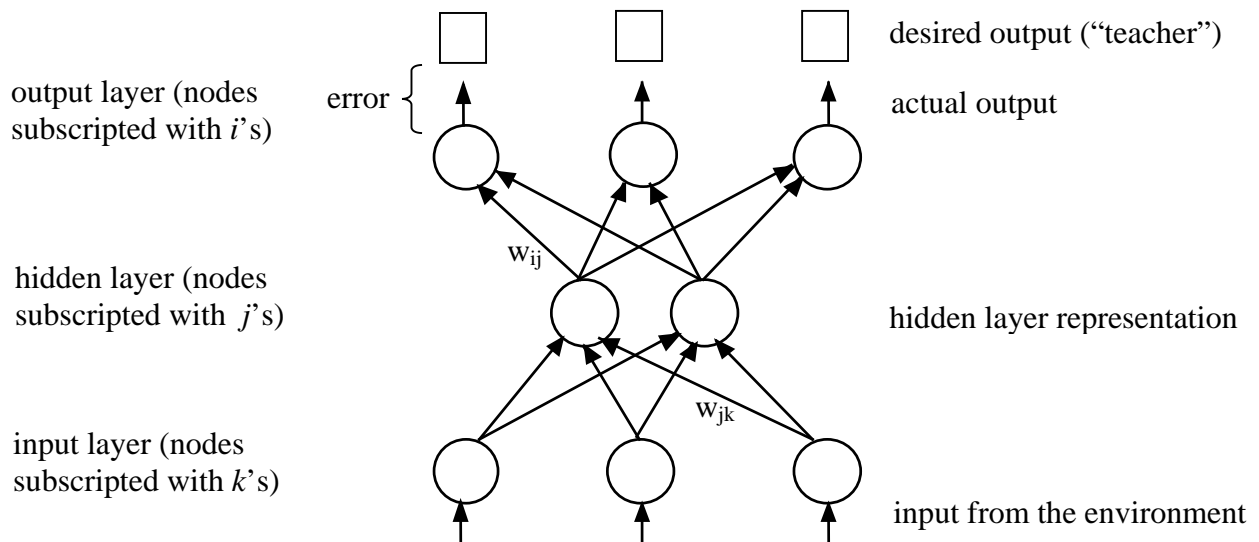
- Unsupervised learning: There is *no error feedback* signal. The network automatically clusters the input into categories. So, for example, if the network is presented with 100 patterns, half of which are different kinds of ellipses and half of which are different types of rectangles, we would expect the network to automatically group these patterns into the two appropriate categories. There is no feedback to tell the network explicitly “this is a rectangle” or “this is an ellipse.”

There are of course many, many variations of specific types of networks within these two major classes. We will consider neural networks from both classes.

Supervised learning: Backpropagation

It turned out that the solution to the linear separability problem posed by Minsky and Papert was the addition of a “hidden” layer between the input layer and the output layer. Although Rosenblatt had discussed multi-layer perceptrons, a weight-change rule for the input-to-hidden units was not found until 1974 by Paul Werbos. This rule was rediscovered (or at least popularized) in Rumelhart et. al (1986), *Parallel Distributed Processing*.

The weight-change technique was called “backpropagation” and is a supervised learning algorithm. Weights in both the hidden-to-output and the input-to-hidden layers are changed based on the error between the network’s actual output and the desired output. Crucially, a “squashing function” was added for the nodes at the hidden and output layers. This sigmoid function continuously “squashed” input to values typically between 0 and 1. This allowed gradient descent to be used as the basis of weight changing throughout the network.



Training of a backpropagation network

The learning of a pattern $P: I \rightarrow T$ involves a feedforward pass in which activation spreads from the input layer to the hidden layer, where it is “squashed” to values between 0 and 1 and then sent on towards the output layer, where it is again squashed before being output by the network. This actual network output (y) is compared to the desired output T over all output nodes and this produces an overall error. This error is backpropagated from the output layer back to the input layer and is used to change the weights in the both of these layers of weights. (See Appendix for the precise gradient descent algorithm.) When the overall error has dropped below a predefined criterion, learning then stops.

When a series of patterns is to be learned by the network, a rather artificial technique called concurrent learning must be used. If each pattern in the series is learned to criterion sequentially, the learning of the new patterns will erase the prior learning of the previous patterns. (This is a serious problem called *catastrophic interference*.) Thus for a series of N patterns, the network must be presented with the:

- 1 epoch {
- 1st pattern, change its weights a little to reduce the error on that pattern;
 - 2nd pattern, change its weights a little to reduce the error on that pattern;
 - etc.
 - last pattern, change its weights a little to reduce the error on that pattern;
 - REPEAT until the error for all patterns is below criterion

Sequence learning

An enormous part of our cognitive learning goes well beyond simple input-output pattern learning. We learn sequences of patterns.

While standard backpropagation networks are fine for learning input-output patterns, they cannot be used effectively to learn sequences of patterns. To understand this, consider the following sequence:

A B C D E F G H I

For this sequence we could train a network to associate the following

A → B
B → C
C → D
D → E
E → F
F → G
G → H
H → I

Then when we gave the network **A** as its “seed”, it would produce **B** on output, which we would feed back into the network to produce **C** on output, and so on. Thus, we could reproduce the original sequence.

But what if the sequence were:

A B C D E F C H I

Unlike the first sequence, we have **C** repeated in the sequence. If we tried to use the technique above, we would have:

A → B
B → C
C → D
D → E
E → F
F → C
C → H
H → I

But the network could not learn this sequence since it has no context to distinguish the two different outputs associated with **C** (for the first occurrence, **D**; for the second, **H**).

We could propose a “sliding window” solution to provide the context. Instead of having the network learn single-letter inputs, if will learn two-letter inputs, thus:

AB → C

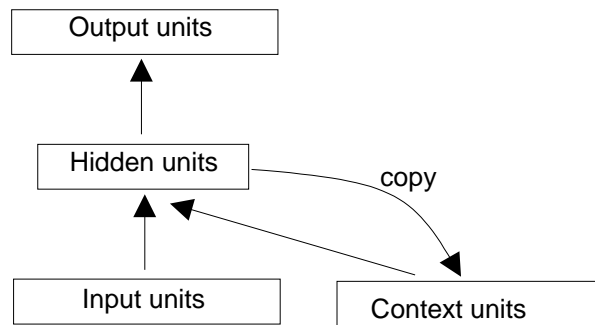
BC → D
CD → E
DE → F
EF → G
FG → H
GH → I

Now the network is fed **AB** as its seed and it can reproduce the sequence with the repeated **C** without difficulty. But what if we need more than one letter's worth of context, as in a sequence like this:

A B C D E B C H I

Now the network needs another context letter...and so on.

Elman (1990) discovered learning algorithm for connectionist networks that did not require the sliding-window technique. The architecture he proposed was the following.



The hidden unit representation from the previous step would be included as the input for the present step. This hidden unit representation for the previous time step is indicated by the bracketed letters:

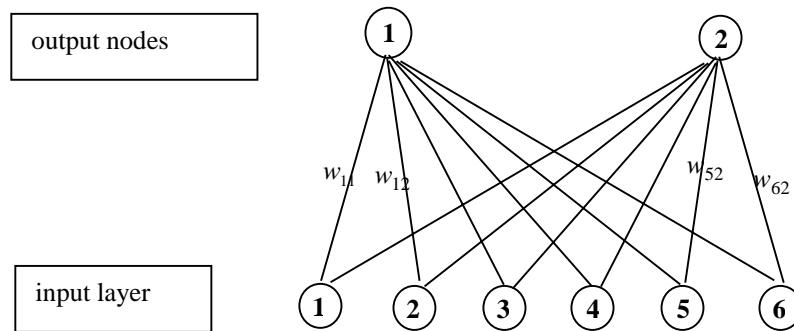
A → B + [no context]
B → C + [A → B]
C → D + [B → C + [A → B]]
D → E + [C → D + [B → C + [A → B]]]
E → F + etc.
F → G + ...
G → H + ...
H → I + ...

This allows the network to continue learning until it has learned the sequence. The more context that is needed will simply mean that there will be a greater number of learning epochs (i.e., learning will take longer).

An example of this type of network is the bilingual memory work for two “toy” languages. See paper included with these notes.

Unsupervised learning:
Kohonen networks

Kohonen networks cluster inputs in a non-supervised manner. The operation of these networks must not be confused with backpropagation type networks where activation spreads along a link to a node, where the activations from all the incoming links to that node are summed. There is no such activation spreading or summing process here. Kohonen networks adjust weight vectors to match input vectors.



There is a metric on the set of output nodes (which is not the case in a standard backpropagation network). In the case illustrated above, there are only two nodes, so there is only one distance, but if there were, say, 25 nodes, we might locate the nodes on a 5x5 grid. We would, in any case, be able to calculate the distance between any two nodes i , and j . We designate this distance as $D(i,j)$.

A vector $V=(v_1, v_2, v_3, v_4, v_5, v_6)$ is presented to our network. V is then compared to $W_1=(w_{11}, w_{21}, w_{31}, w_{41}, w_{51}, w_{61})$ and $W_2=(w_{12}, w_{22}, w_{32}, w_{42}, w_{52}, w_{62})$. Whichever weight vector is closest to is designated as the central weight vector. We can call it W_c . The weights of W_c are modified so as to more closely resemble the values of the input vector V , thus: $\Delta w_{ij} = \eta (v_j - w_{ij})$, where η is the learning rate. For the other weight vectors, we include their distance from the central vector in the weight change calculation. The further they are from it, the less their weights will change, thus: $\Delta w_{ij} = \eta D(i, j)(v_j - w_{ij})$

In this way, the weights of the network will gradually come to correspond to the “prototypical” vectors of the category associated with their particular output node. Thus, when a new vector is presented to the network, it will be most resemble the set of set of network weights that were formed by previous input vectors most resembling the new vector.

Conclusions

Our goal has been to very briefly examine neural networks in the context of gaining a better understanding of how our own cognitive processes might work. Although the history of connectionism arguably can be traced to William James in the late 19th century, we have chosen as our starting point McCulloch and Pitts 1943 paper on “neuro-logic” and Hebb’s landmark book that attempted to link the neural level to the molar level of cognition through his ideas of a synaptic learning rule and the development of cell

assemblies in the brain. We studied the direct outgrowth of these ideas, the perceptron, its demise and its reappearance as the multi-layered perceptron.

The next frontier will undoubtedly be in computational neuroscience where, instead of networks of (slightly modified) McCulloch-Pitts neurons, we will use spiking neurons, and variables such as their connection density, their firing timing and synchrony, and so on. We are almost at a point where the population dynamics of large networks of these kinds of simulated neurons can realistically be studied. And even further in the future neuronal models with even more detail, relying on the Hodgkin-Huxley equations of membrane potentials and neuronal firing, can be incorporated into our models. With ever more accurate models, we will be able to develop ever more accurate predictions of how real neuronal systems work. Gradually, neural network models will become good enough to interact with neurobiology and neuropsychology in a real manner, much as the experimental and theoretical branches of all of the hard sciences interact.