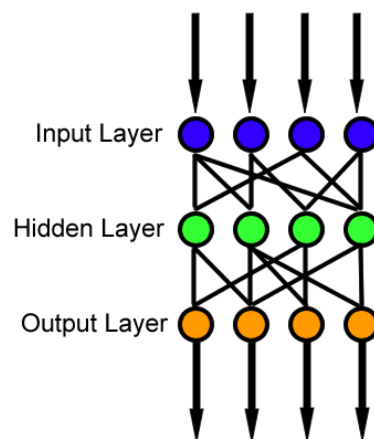


1 Feed-Forward Networks

In the Feed-forward networks, the information moves in only one direction (forward) from the input nodes, through the hidden nodes (if they exist), and to the output nodes. **There are no cycles in this network.**

- Multiple layers of neurons with a certain number of inputs and outputs.
- Every layer of nodes feeds the next layer with inputs.
- There is an input and an output layer with hidden layers in between.



A single unit, like a perceptron, can be seen as a feed-forward network.

In feed-forward networks it is easy to evaluate each unit. The outputs are continuous functions of the input, which facilitates the optimization in case of wrong outputs. The training can be done using "training data": input/output pairs (x_i, d_i) . Where x_i is the input value and d_i is the desired output. Then we can define the error $E = \sum_k (f(x_k) - d_k)^2$, where $f(x_k)$ is the output of the network. How can we change the weights to reduce the error? We can use gradient descent. We calculate all the $\frac{\partial E}{\partial w_i}$ in the network, easily, by starting at the end and then walking backwards. This is known as Backpropagation.

1.1 Boltzmann Machines

Boltzmann Machines were invented in 1985 but not by Boltzmann. The name is given because these units use a Boltzmann distribution in their sampling function.

These units are similar to Hopfield Networks, however, they have a probability of being active. When updating a unit, we set its value to zero or one probabilistically, following a sampling function, see Figure 1. Boltzmann machines do not converge, they do not reach a stable state. It can be seen as a system for sampling. It's a way to do a random walking in the state space.

Sampling There are many types of distributions. When we want to get examples of these distributions, we need to sample from it. And giving some samples we can recovery the distribution of the data.

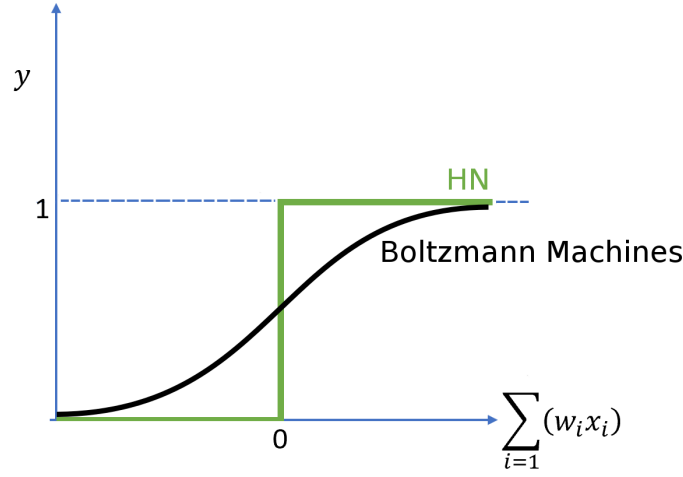


Figure 1: Representations of activation function for Hopfield Networks (HN) and Boltzmann Machines

We want the units to forget previous states so the sampling is not biased, i.e., not similar to previous ones, thus really "random". While this sampling is nice, it is not useful as a memory, i.e., we don't want to sample things randomly from our memory.

We can define a network where the units have a real-valued activity level ($a_i \in [0, 1]$), and also we can make time continuous, so $\frac{\partial a_i}{\partial t} = \theta(\Sigma) - a_i$.

Using units like this, we can make a feed-forward network.

1.2 Backpropagation and Error function

Backpropagation is the process of calculating the derivatives, using the chain rule, from the last layer (directly connected with the output, thus, with the loss function) to the first layer (connected with the inputs). This process can be seen as walking through the network in a backwards manner.

- The inputs and desired outputs are given as $S = \{(x, d)^1, \dots, (x, d)^l\}$.
- The error function is given as $E(S) = \sum_i \frac{1}{2} \|y(x^i) - d^i\|^2$.
- The output is a non-linear transformation $y = f(a)$.
- $f(a)$ is the activation function, which is usually a sigmoid function.
- The error function for a single training sample is $E(S) = \frac{1}{2} (f(x_1 w_1 + x_2 w_2 + w_0) - d)^2$.
- The output of a simple network is for example $y(x_1, x_2, x_3) = f(x_1 w_{21} + f(x_2 w_{11} + x_3 w_{12} + w_{10}) w_{22} + w_{20})$.
- $\frac{\partial E(w_1, w_2, w_0)}{\partial w_1} = (f(x_1 w_1 + x_2 w_2 + w_0) - d) \cdot f'(x_1 w_1 + x_2 w_2 + w_0) \cdot x_1$.
- $\frac{\partial E(w_1, w_2, w_0)}{\partial w_2} = (f(x_1 w_1 + x_2 w_2 + w_0) - d) \cdot f'(x_1 w_1 + x_2 w_2 + w_0) \cdot x_2$.
- $\frac{\partial E(w_1, w_2, w_0)}{\partial w_0} = (f(x_1 w_1 + x_2 w_2 + w_0) - d) \cdot f'(x_1 w_1 + x_2 w_2 + w_0)$.
- The error terms travel backwards through the network and get multiplied with the derivative of the activation function of that input. Multiple error terms can just be added up.
- The partial derivative of the error E term in relation to the weight w to be adjusted can be added to the weight in order to learn. An additional weighting factor can be added.

1.3 Gradient Descent

Consider $E = \sum (f_w(x) - y_i)^2$, we want to adjust \vec{w} (weights of the network) to minimize E . Gradient descent is the process of descending through the gradients (using the derivatives calculated with backpropagation), in this algorithm we try to reach the minimum of the loss function. This is an iterative process.

Generally, the iterative process is given by $\theta_{inew} = \theta_i + \Delta\theta_i$, where $\Delta\theta_i = -\alpha \frac{\partial J(\theta)}{\partial \theta_i}$.

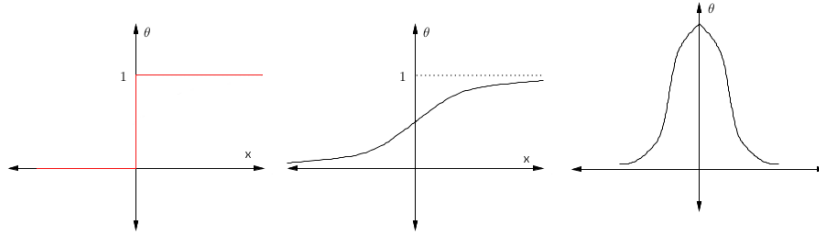


Figure 2: left-most figure: $f_{w_0}(x) = \theta(wf + w\dots)$, $\frac{df}{dw} = 0 \rightarrow \theta$ is not a good threshold function. To know in what direction we should move to find our minima, we need to use a threshold function that is continuous and differentiable, like the one in the center figure. Right figure: $\frac{df}{dw} = \theta'$.