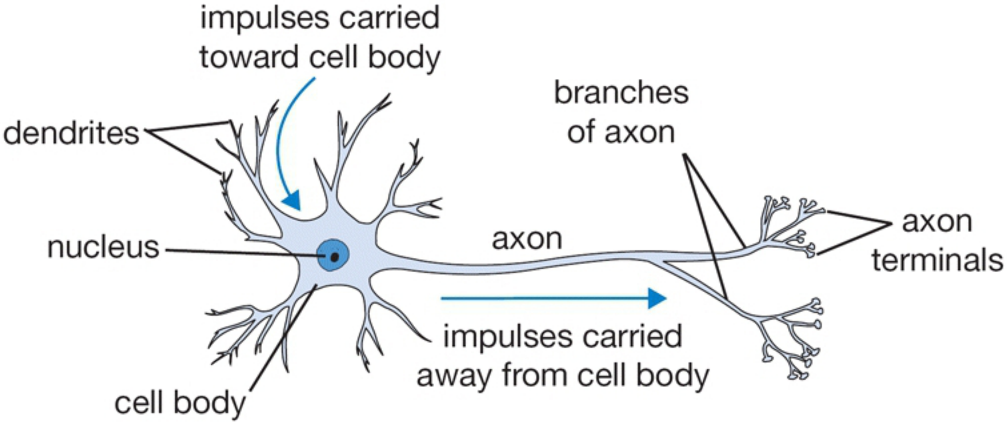
Deep Learning

Paolo Bettelini

Contents

1 Types of neurons

1.1 Brain neurons


impulses carried toward cell body
dendrites
branches of axon
axon terminals
nucleus
axon
impulses carried away from cell body
cell body

1.2 Linear neurons

A linear neuron is very simple and computationally limited in what it can do.

$$y = b + \sum_i x_i w_i$$

The output y is given by the bias b plus the sum of all the input connections x_i multiplied by their weight w_i.

1.3 Binary threshold neurons

Binary threshold neurons output a 1 or a 0 depending on its weighted value.

Given a threshold $\theta = -b$

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} 1 \text{ if } z \geq 0 \\ 0 \text{ otherwise} \end{cases}$$

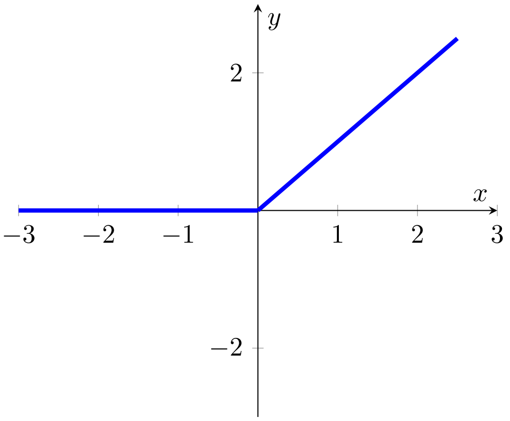
1.4 Rectified Linear Neurons or Linear threshold neurons

They compute a linear weighted sum of their inputs.
The output is a non-linear function of the total input.

Given a threshold $\theta = -b$

$$z = b + \sum_i x_i w_i$$

$$y = \begin{cases} z \text{ if } z > 0 \\ 0 \text{ otherwise} \end{cases}$$



1.5 Sigmoid neurons

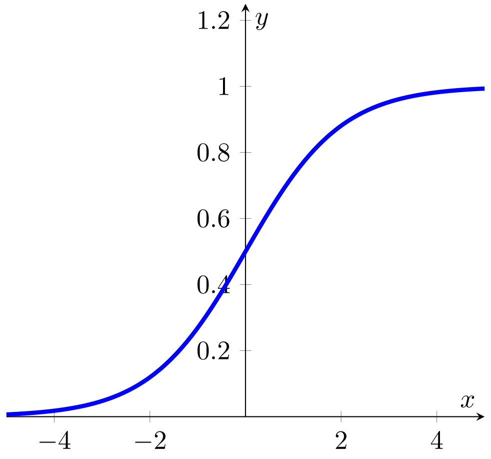
They give a real-valued output that is a smooth and bounded function of their total input.

The logistic function is often used.

Given a threshold $\theta = -b$

$$z = b + \sum_i x_i w_i$$

$$y = \frac{1}{1 + e^{-z}}$$



This function has smooth derivatives that change continuously.
This characteristic makes the learning process easier.

2 Types of learning

2.1 Supervised learning

Each training consists of making the network guess the target output t for a certain input x, given the difference between the correct target and the guess we tweak the network.

There are two type of supervised learning

2.1.1 Regression

The target output is a numeric value of a vector of values.
To describe the error we can compute the square difference between the target output t and the actual output y of the module.
Often the value $\frac{1}{2}(t-y)^2$ is used. The $\frac{1}{2}$ coefficient is there to cancel the 2 out when differentiation is applied.

2.1.2 Classification

The target output is a class or label. Usually either 1 or 0. There could also be multiple labels. [...]

2.2 Reinforcement learning

The output is an action of a sequence of actions to maximize the sum of rewards.

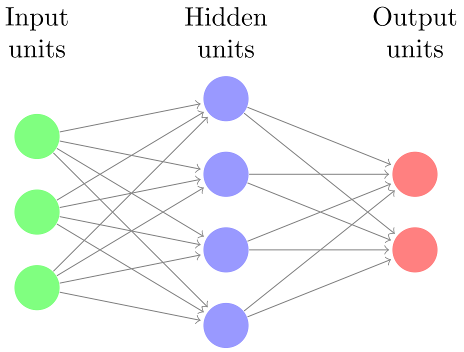
2.3 Unsupervised learning

[...]

3 Types of network architecture

3.1 Feed-forward neural network

The most common type of neural network is the feed-forward neural network.
If there are more than one hidden layer it is called a *deep* neural network.



At each layer there is a different representation of the initial input.
Similar things may become less similar and less similar things may become more similar.

In order to achieve this each layer must be a non-linear function of the activities in the previous layer.

3.2 Recurrent network

The same input might pass multiple times through the same neuron.
They are very difficult to train and more biologically realistic, hence they are more powerful.

Recurrent networks have the ability to remember information in their hidden state.

3.3 Symmetrically connected network

They are like recurrent network, but the connections between units are symmetrical (they have the same weight in both directions).

Much easier to analyze than recurrent networks, but are more restricted in what they can do.

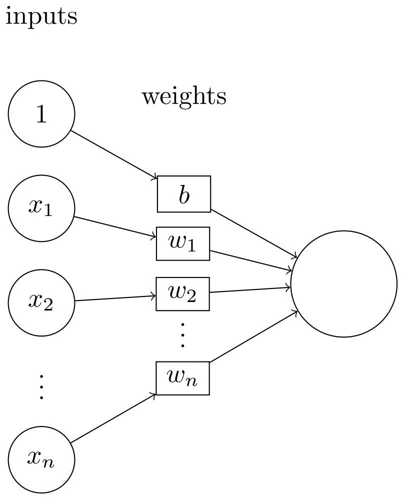
Symmetrically connected networks without hidden units are called *Hopfield networks*.

4 The perceptron

A perceptron is an example of a statistical pattern recognition system.
The decision unit in a perceptron is a binary threadhold neuron.

4.1 Learning biases

We can learn the bias by treating it just like a regular weight, rather than having a separate learning rule.
The bias is just the wight of an extra feature with the value of 1.



4.2 Convergence procedure

- Add an extra component with value 1 to each input vector. This serves as the bias and is equivalent to the negative of the threshold.
- Pick training cases such that every training case will keep getting picked.
  - If the output unit incorrectly outputs a 0, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a 1, subtract the input vector to the weight vector.
  - If the output unit is correct, don't do anything.

4.3 Geometric view

Each training case defines a hyperplane (depending on the number of components) where for an *input vector* there is a *weigth vector* that can be either on the correct or wrong side of the hyperplane, depending on the resulting scalar product. The plane goes through the origin and is perpendicular to the input vector.

Considering multiple training cases, the resulting vector shall be on all the good sides of all the hyperplanes. Such point may not exist. However, if such point exists, the convergence procedure is guaranteed to find it.

4.4 Convergence

A region in the hyperspace where all the vectors in the regon satisfy all the training case exists if the dataset is linearly separable.
This means that for example, considering a two-dimensional binary input vector, a perceptron cannot tell if the inputs are the same (the XOR problem). This would lead to four inequalities that are impossible to satisfy.

5 Linear neuron

5.1 Definition

The perceptron learning procedure cannot extended to an arbitrary number of hidden layers.

The simplest example is a linear neuron with a squared error measure. The goal is to minimize the errore over all training cases, rather than the weight vector to be in a given region. The output of a linear neuron is a real value output which is a weighted sum of its input.

$$y = \sum_i w_i x_i$$

For each training case we tweak each weight

$$\Delta w_i = \epsilon x_i (t - y)$$

where ϵ is some *learning rate*, y is the guess of the network and t is the correct target output.

There is no guarantee with this kind of learning that the single weights will get better.

5.2 The delta rule

We can define the error of the network as the squared residuals summed over all training cases

$$E = \frac{1}{2} \sum_{n \in \text{traning}} (t_n - y_n)^2$$

Now differentiate to get error derivatives for weights. The purpose of the $\frac{1}{2}$ term is to get a nice differentiation.

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{1}{2} \sum_n \frac{\partial y_n}{\partial w_i} \frac{dE_n}{dy_n} \\ &= - \sum_n x_{ni} (t_n - y_n) \end{aligned}$$

How the error changes as we change the weights, will be how the output changes as we change the weights times how the error changes as the change the output.

So now the change of a weight is

$$\Delta w_i = -\epsilon \frac{\partial E}{\partial w_i} = \epsilon \sum_n x_{in} (t_n - y_n)$$

6 Logistic neuron

6.1 Learning weights

We can generalize the learning rule for a linear neuron to a logistic neuron, which is a non-linear neuron.

$$z = b + \sum_i x_i w_i$$
$$y = \frac{1}{1 + e^{-z}}$$

First we need to compute the derivatives of z with respect to the inputs x_i, weights w_i and the output y.

$$\frac{\partial z}{\partial w_i} = x_i, \quad \frac{\partial z}{\partial x_i} = w_i, \quad \frac{dy}{dz} = y(1-y)$$

Now, using the chain rule, we can compute the derivative of the output with respect to w_i

$$\frac{\partial y}{\partial w_i} = \frac{\partial z}{\partial w_i}\frac{dy}{dz} = x_i y(1-y)$$

Which means that the error E changes with respect to w_i as

$$\frac{\partial E}{\partial w_i} = \sum_n \frac{\partial y_n}{\partial w_i}\frac{\partial E}{\partial y_n} = -\sum_n x_{in} y_n (1-y_n)(t_n - y_n)$$

Therefore the learning rule for a logistic neuron is

$$\Delta x_i = -\epsilon \frac{\partial E}{\partial w_i} = \epsilon \sum_n x_{in} y_n (1-y_n)(t_n - y_n)$$