Neural networks

The main motivation was to mimic the learning of biological networks, such as the human brain

Perceptron

Is a simple decision rule.

For instance, you want to decide whether you are going to the store or not on a particular day. We suppose that this decision is based on the occurrence of 3 different events x1, x2 and x3 as follows.

To each event, we can assign a weight saying how important it's in our decision.

event	weight
x_1 = the sun shines	$w_1 = 1.1$
x_2 = you are out of milk	$w_2 = 0.2$
x_3 = you have a valid coupon	$w_3 = 0.8$

The classification on whether you will go to the store depends of whether a certain number of the event occur, for example,

$$\text{if and only if} \quad \sum_i w_i x_i = \mathbf{w}^T\mathbf{x} \geq 1$$

then I'll go to the store.

We can remove the threshold by introducing a bias term b, which in this case gives b = -1, yielding the decision boundary

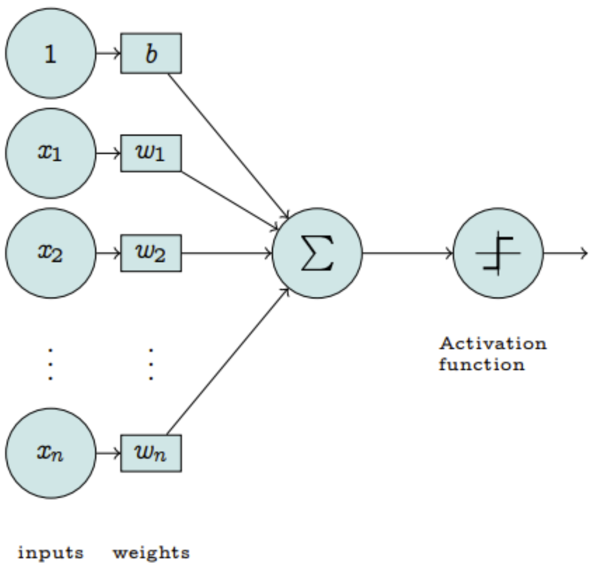
$$\sum_i w_i x_i + b = \mathbf{w}^\intercal \mathbf{x} + b \geq 0.$$

Perceptron is a basic computational unit of artificial neural networks and serves as a building block for more complex neural network architectures. A perception takes one or multiple inputs, applies weights to these inputs, computes a weighted sum, and passes the sum through an activation function to produce an output.

Main components of a perceptron:

1. Inputs: each input represents a feature or attribute of the input data.
2. Weights: each input is associated with a weight that determines the importance of that input in the computation. The weights can be positive or negative.
3. Weighted Sum: the perception calculates the weighted sum of the inputs by multiplying each input with its corresponding weight and summing them up.
4. Activation Function: The weighted sum is passed through an activation function. which introduces non-linearity and determines the output of the perceptron.

The training of perceptions involves adjusting the weights based on the error between the predicted output and the desired output. This process is typically done using a learning rule called the perception learning rule or delta rule.



Perceptron Learning

The classical training procedure of the perception is done as follows:

For each misclassified element x we perform the following update on the weights:

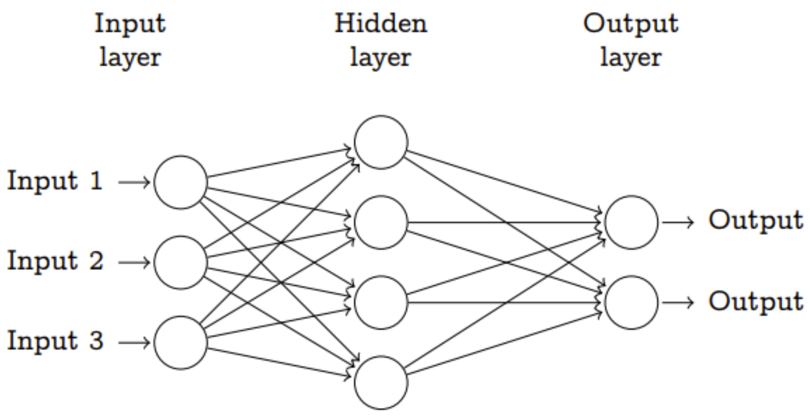
$$\mathbf{w} \leftarrow \mathbf{w} + \eta(y - \hat{y})\mathbf{x}$$

where n > 0 is the learning rate.

Multilayer Perceptron to Feed Forward Neural Network

Feed Forward Neural Networks are obtained by combining several neurons, or composing several perceptrons.

A neural network consists of several layers, the input layer, the hidden layers and the output layer.



wij → i represents the index that is going to, j represents the index that comes from

w12 ⇒ is going to the first neuron, and coming from the second

Neural networks learn their own features

One particular strength of NN is its ability to learn the features.

The learning objective with neural networks is to learn the weights and the biases, which minimize some cost functions. Hence, in clear terms, we want to minimize some costs J(W,b).

This is in many cases fairly non-trivial and definitely non-convex. However, as it turns out one of the standard ways to go is by using gradient descent.

Butttt is slow when the number of training examples increases. Solution: stochastic gradient descent.

The key idea of stochastic gradient descent: compute the gradient using a random sample of the training data. The assumption is that this will be a good enough approximation to improve the results.

Overfitting

Neural networks sometimes perform too well. Hence, they are prone to overfit.

Some ideas to prevent overfitting:

Regularization

Early Stopping

Dropout

Reduce the number of neurons, and/or layers