Introduction to Neural Networks

Reading Assignment

Wikipedia Articles on Neural Networks and Reinforcement Learning

Neural Networks

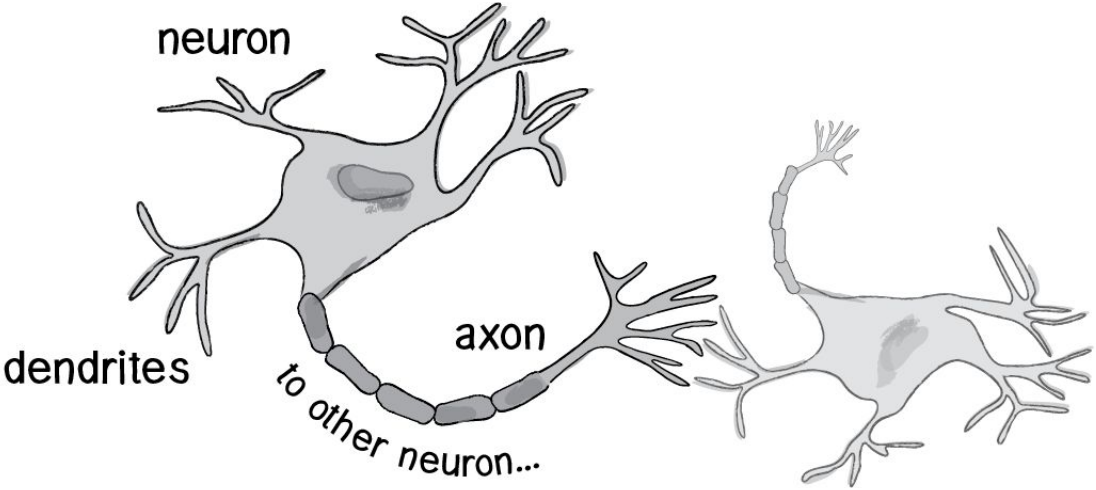
Neural networks are modeled after biological neural networks and attempt to allow computers to learn in a similar manner to humans - reinforcement learning.

Use cases:

- Pattern Recognition
- Time Series Predictions
- Signal Processing
- Anomaly Detection
- Control

Neural Networks

The human brain has interconnected neurons with dendrites that receive inputs, and then based on those inputs, produce an electrical signal output through the axon.



Neural Networks

There are problems that are difficult for humans but easy for computers (e.g. calculating large arithmetic problems)

Then there are problems easy for humans, but difficult for computers (e.g. recognizing a picture of a person from the side)

Neural Networks

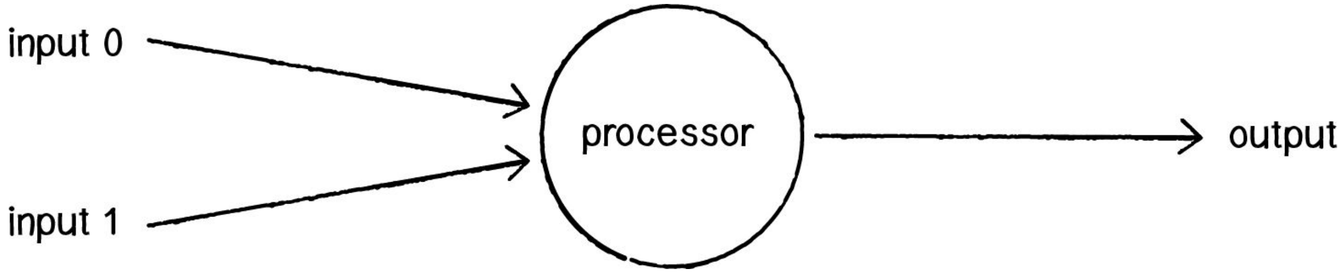
Neural Networks attempt to solve problems that would normally be easy for humans but hard for computers!

Let's start by looking at the simplest Neural network possible - the perceptron.

Perceptron

A perceptron consists of one or more inputs, a processor, and a single output.

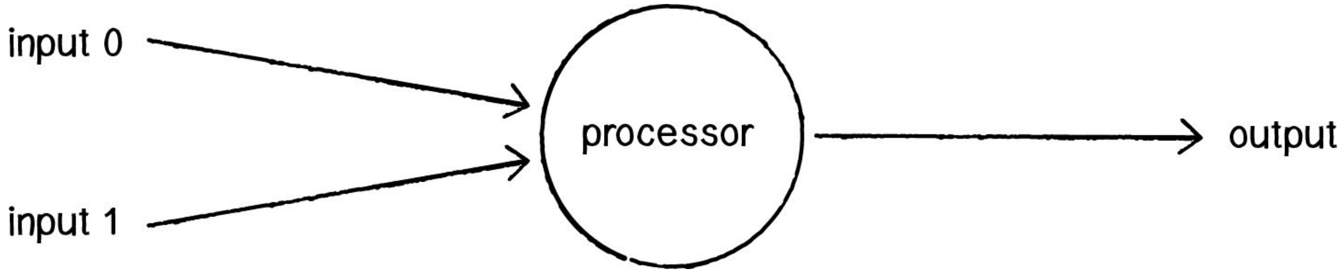
A perceptron follows the "feed-forward" model, meaning inputs are sent into the neuron, are processed, and result in an output.



Perceptron

A perceptron process follows 4 main steps:

1. Receive Inputs
2. Weight Inputs
3. Sum Inputs
4. Generate Output

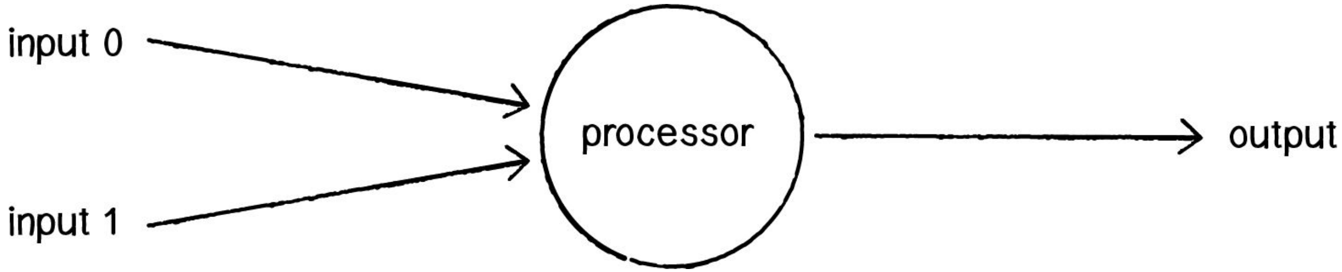


Perceptron

Say we have a perceptron with two inputs:

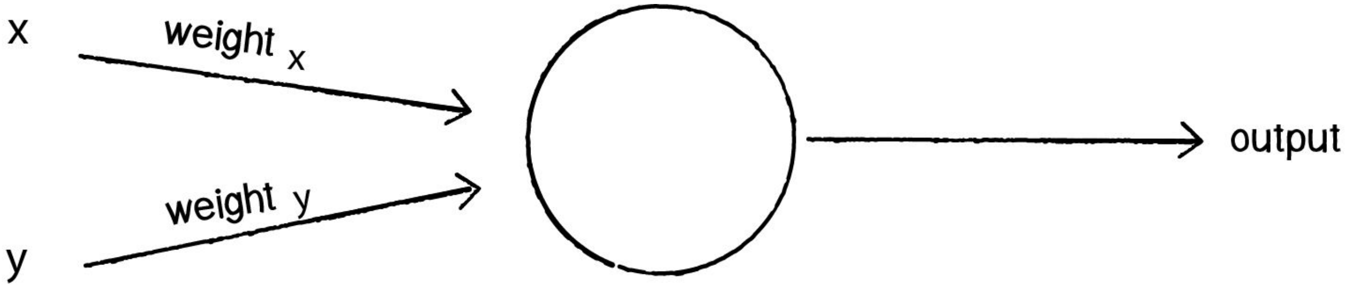
Input 0: x1 = 12

Input 1: x2 = 4



Perceptron

Each input that is sent into the neuron must first be weighted, i.e. multiplied by some value (often a number between -1 and 1).

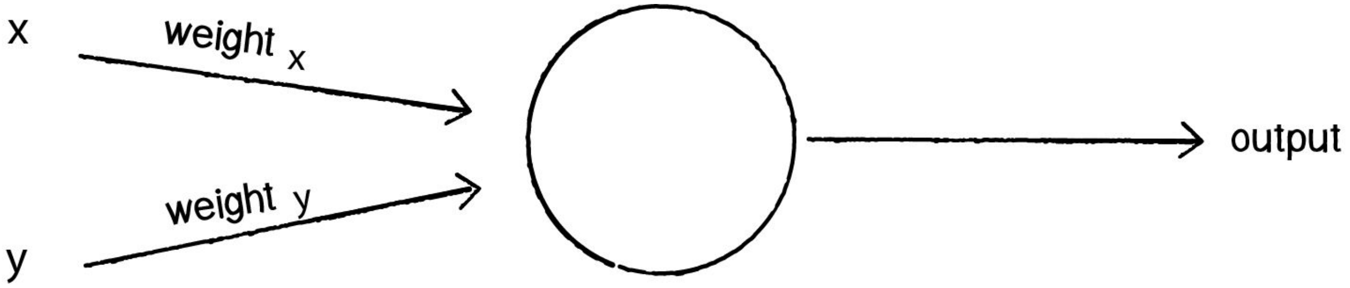


Perceptron

When creating a perceptron, we'll typically begin by assigning random weights.

Weight 0: 0.5

Weight 1: -1

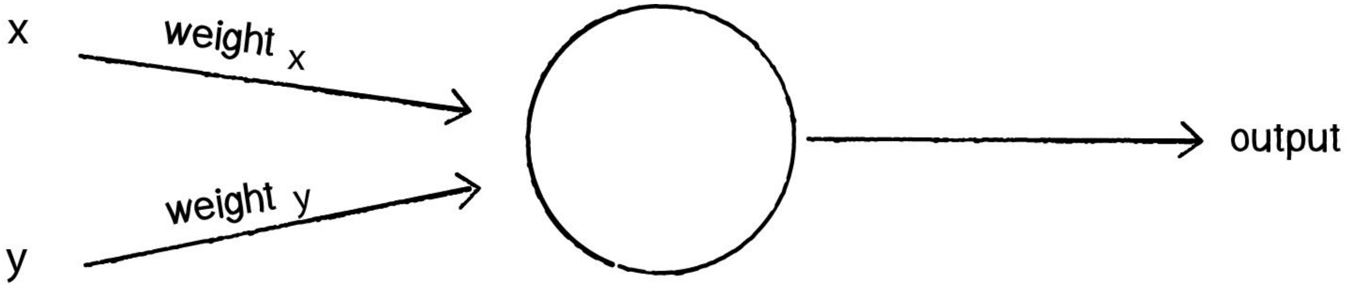


Perceptron

We take each input and multiply it by its weight.

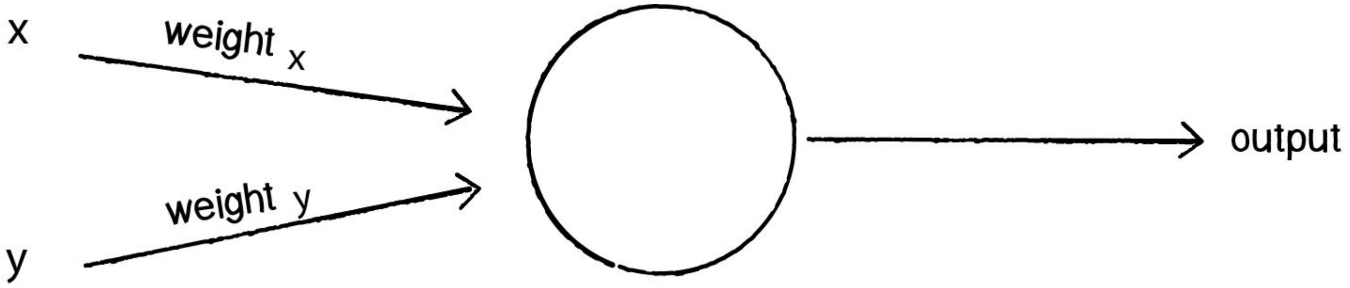
Input 0 * Weight 0 ⇒ 12 * 0.5 = 6

Input 1 * Weight 1 ⇒ 4 * -1 = -4



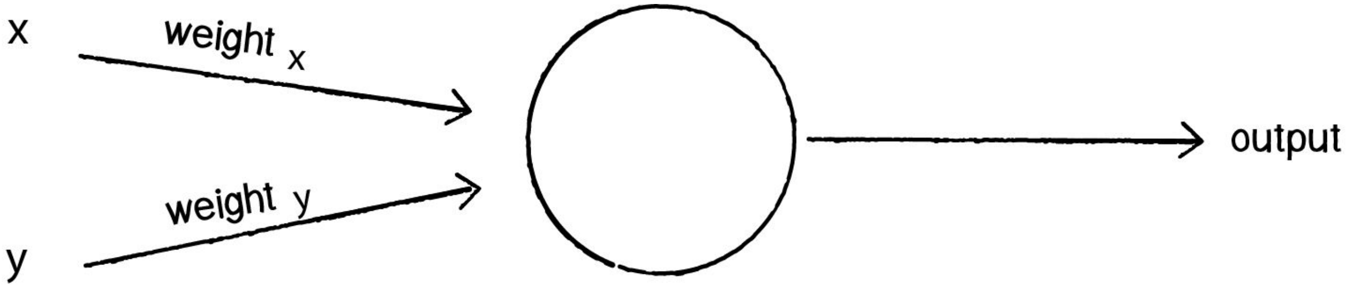
Perceptron

The output of a perceptron is generated by passing that sum through an activation function. In the case of a simple binary output, the activation function is what tells the perceptron whether to “fire” or not.



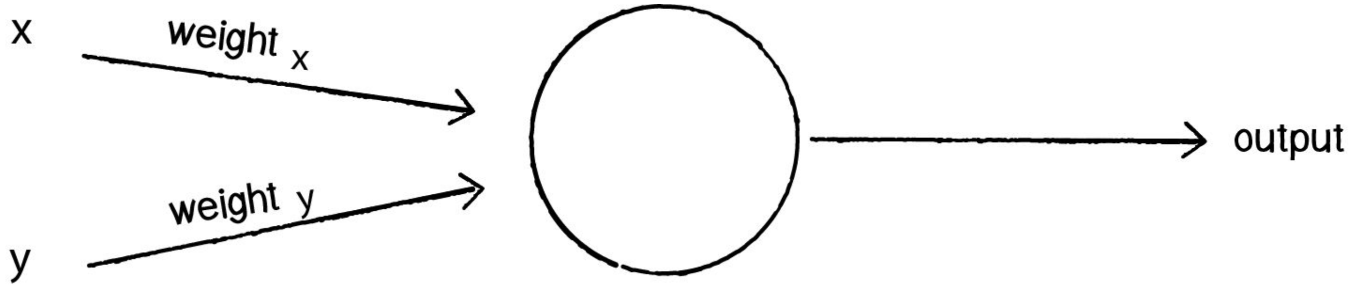
Perceptron

Many activation functions to choose from (Logistic, Trigonometric, Step, etc…). Let’s make the activation function the sign of the sum. In other words, if the sum is a positive number, the output is 1; if it is negative, the output is -1.



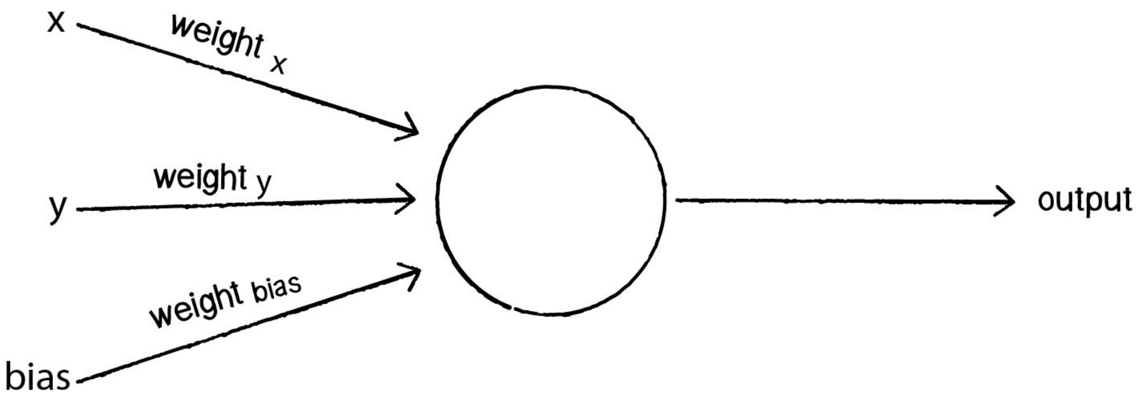
Perceptron

One more thing to consider is Bias. Imagine that both inputs were equal to zero, then any sum no matter what multiplicative weight would also be zero!



Perceptron

To avoid this problem, we add a third input known as a bias input with a value of 1. This avoids the zero issue!



Perceptron

To actually train the perceptron we use the following steps:

1. Provide the perceptron with inputs for which there is a known answer.
2. Ask the perceptron to guess an answer.
3. Compute the error. (How far off from the correct answer?)
4. Adjust all the weights according to the error.
5. Return to Step 1 and repeat!

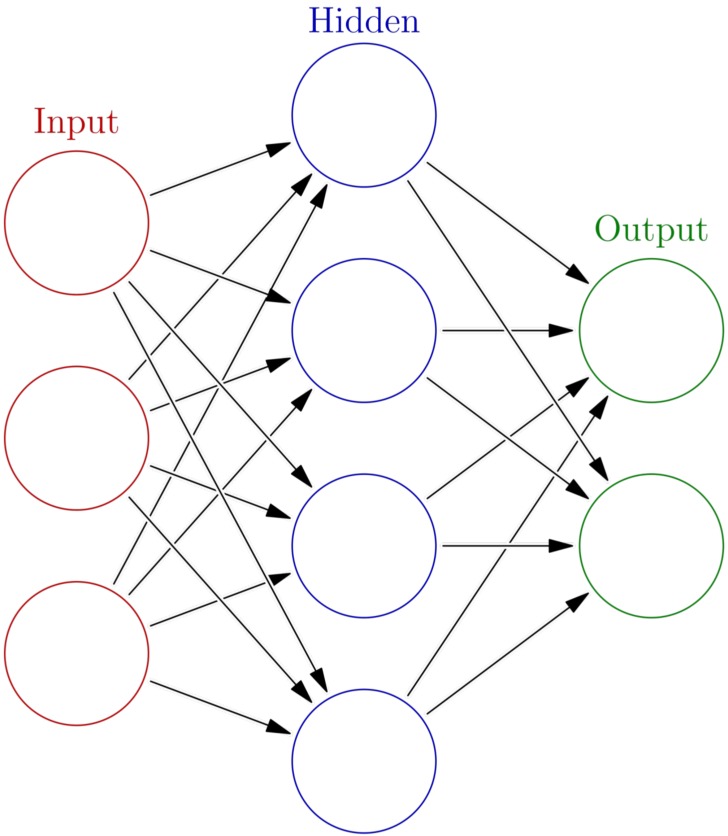
Perceptron

We repeat this until we reach an error we are satisfied with (we set this before hand).

That is how a single perceptron would work, now to create a neural network all you have to do is link many perceptrons together in layers!

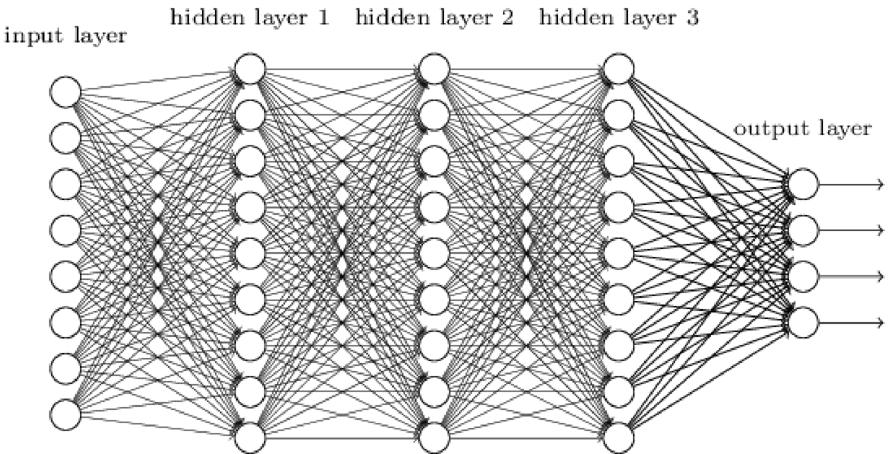
Neural Networks

You'll have an input layer and an output layer. Any layers in between are known as hidden layers, because you don't directly "see" anything but the input or output.



Neural Networks

You may have heard of the term “Deep Learning”. That’s just a Neural Network with many hidden layers, causing it to be “deep”. For example, Microsoft’s state of the art vision recognition uses 152 layers.



Example with R

Let's go to RStudio and begin to explore an example, then you’ll have a project to test your understanding!