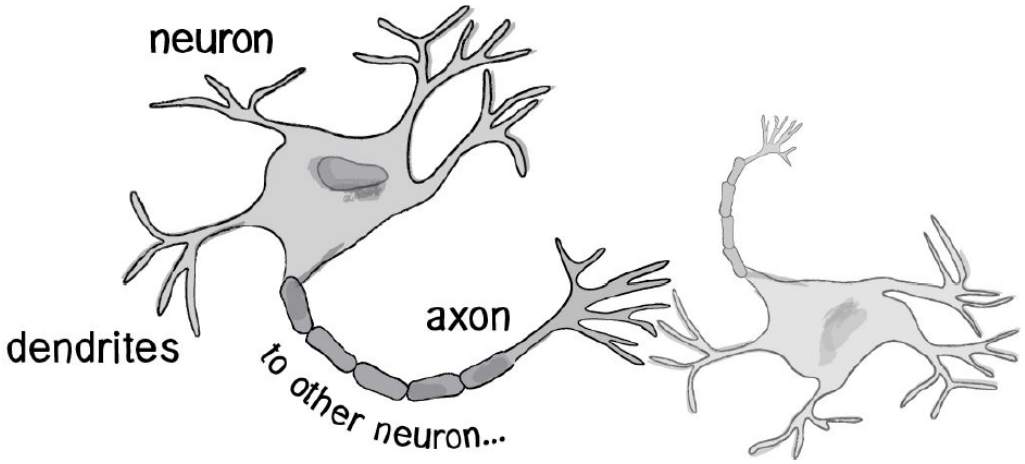
TensorFlow

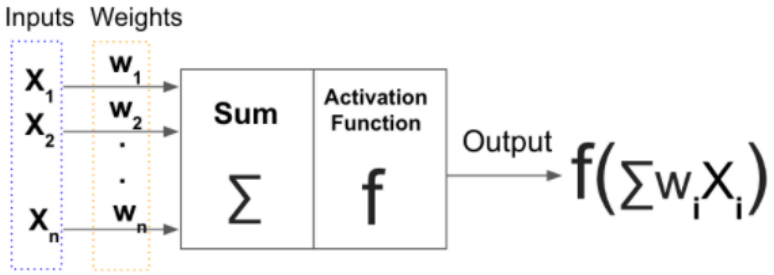
Neural Networks

- 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

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.





Structure of artificial neuron

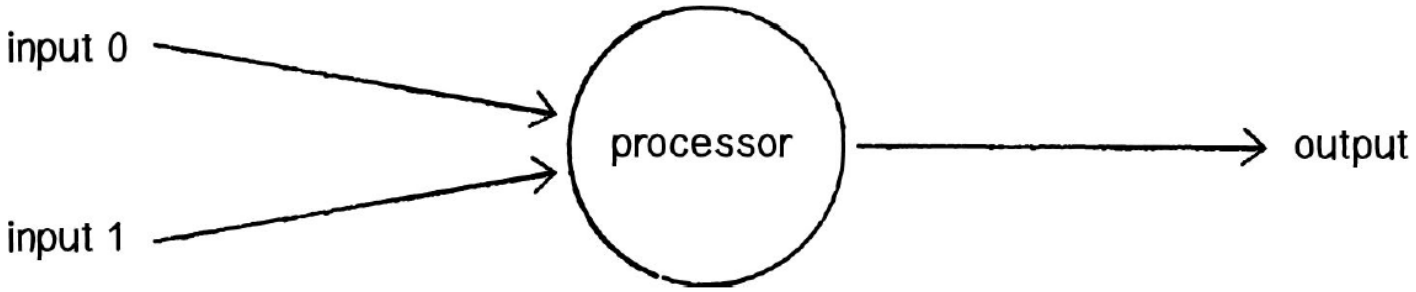
Neural Networks

- 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 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.
- A perceptron process follows 4 main steps:
  1. Receive Inputs
  2. Weight Inputs
  3. Sum Inputs
  4. Generate Output



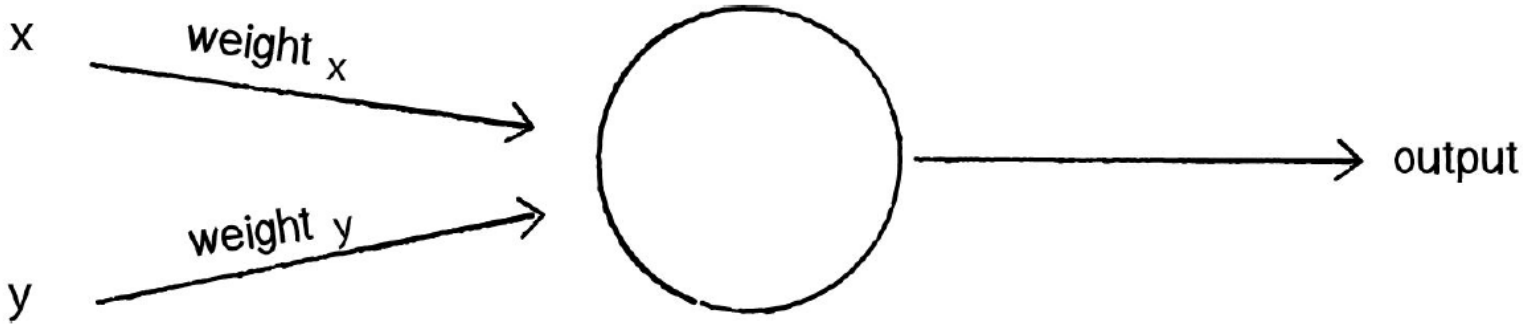
Perceptron

- Say we have a perceptron with two inputs:

Input 0: x1 = 12

Input 1: x2 = 4

- 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

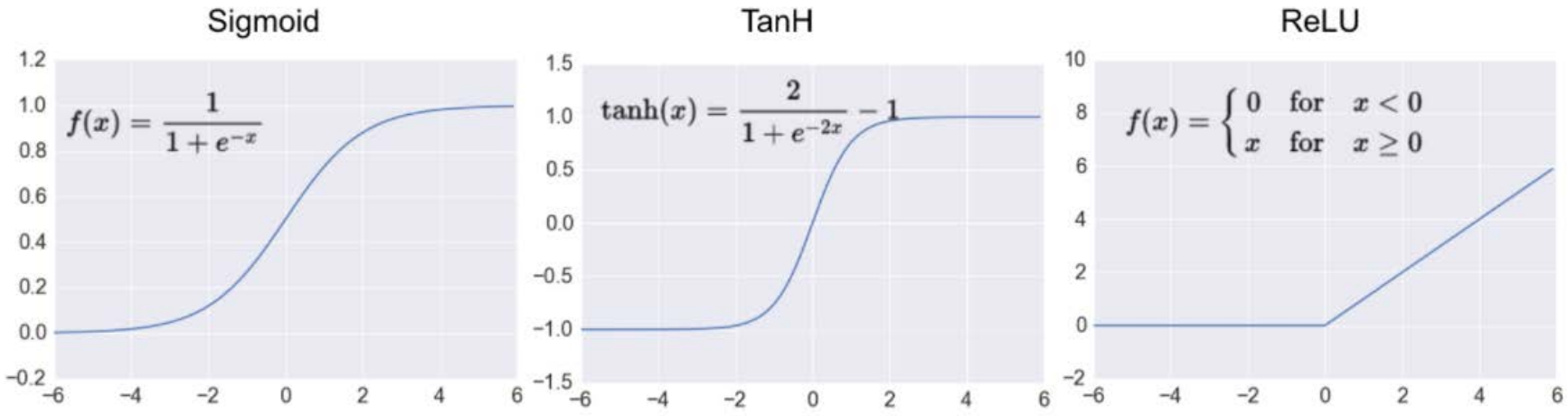
- We take each input and multiply it by its weight.

Input 0 * Weight 0 ⇒ 12 * 0.5 = 6

Input 1 * Weight 1 ⇒ 4 * -1 = -4

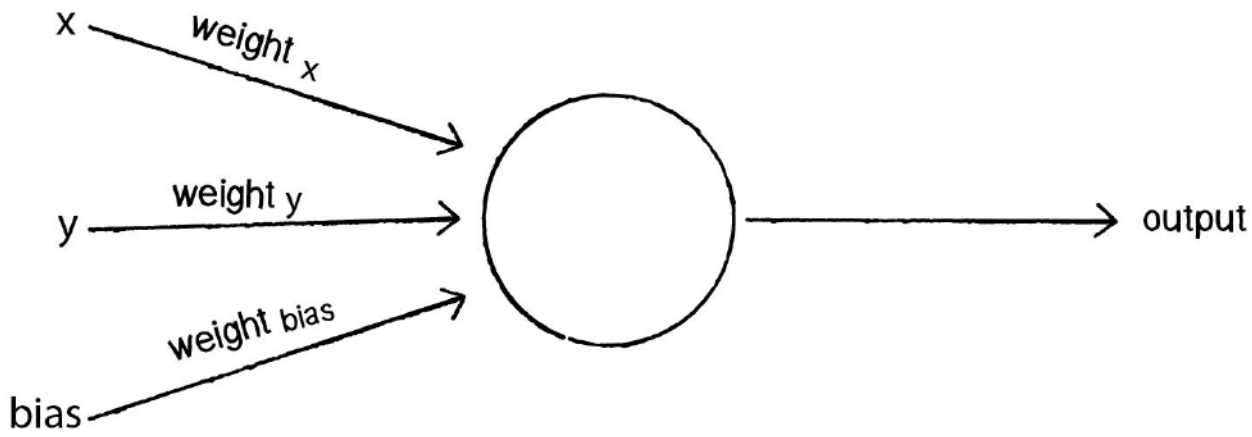
Perceptron – Activation Function

- 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 – Bias

- 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

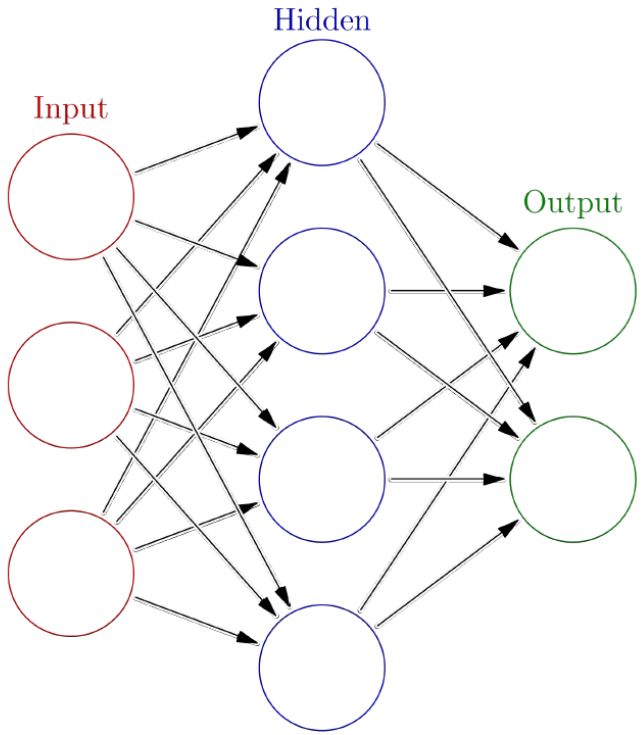
- To actually train the perceptron we use the following steps:

  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!

- 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 perceptron's together in layers!

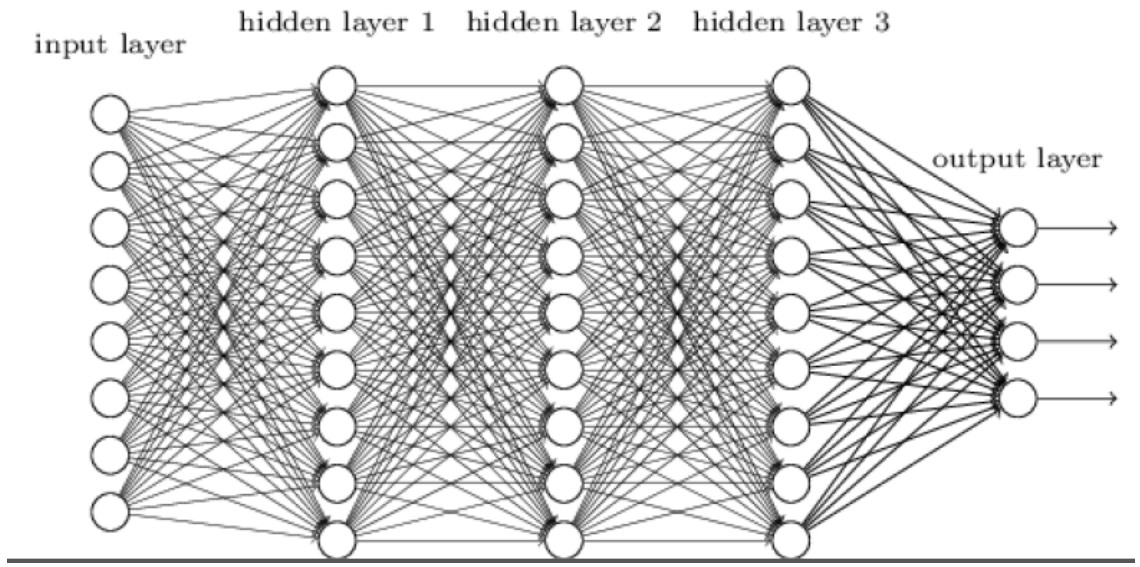
Neural Networks

- 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.



Deep Learning

- “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.



TensorFlow

- TensorFlow is an open source software library developed by Google
- It has quickly become the most popular Deep Learning Library in the field
- It can run on either CPU or GPU
- Typically, Deep Neural Networks run much faster on GPU
- The basic idea of TensorFlow is to be able to create data flow graphs.
- These graphs have nodes and edges
- The arrays (data) passed along from layer of nodes to layer of nodes is known as a Tensor

TensorFlow

- There are two ways to use TensorFlow
    1. Customizable Graph Session
    2. SciKit-Learn type interface with Contrib.Learn

More on https://www.tensorflow.org/