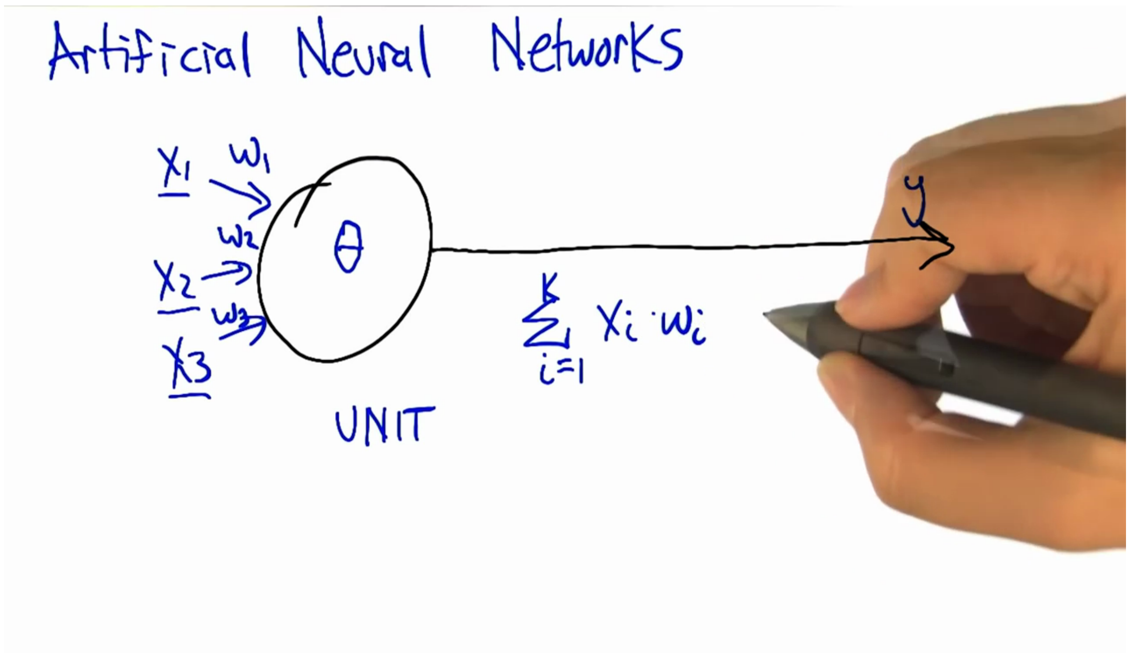
## Learning and Loss

Like `MiniFlow` in its current state, neural networks take inputs and produce outputs. But unlike `MiniFlow`in its current state, neural networks can *improve* the accuracy of their outputs over time (it's hard to imagine improving the accuracy of `Add` over time!). To explore why accuracy matters, I want you to first implement a trickier (and more useful!) node than `Add`: the `Linear` node.

## The Linear Function



As described by Charles and Michael, a Neuron calculates the weighted sum of its inputs.

Think back to Neural Networks lesson with Luis and Matt. A simple artificial neuron depends on three components:

- inputs, *x* (vector)
- weights, *w*(vector)
- bias, *b* (scalar)

The output, *o*, is just the weighted sum of the inputs plus the bias:

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

Equation (1)

Remember, by varying the weights, you can vary the amount of influence any given input has on the output. The learning aspect of neural networks takes place during a process known as backpropagation. In backpropogation, the network modifies the weights to improve the network's output accuracy. You'll be applying all of this shortly.

In this next quiz, you'll try to build a linear neuron that generates an output by applying a simplified version of Equation (1). `Linear`should take an list of inbound nodes of length *n*, a list of weights of length *n*, and a bias.

## Instructions

1. Open nn.py below. Read through the neural network to see the expected output of `Linear`.
2. Open miniflow.py below. Modify `Linear`, which is a subclass of `Node`, to generate an output with Equation (1).

(Hint: you could use `numpy` to solve this quiz if you'd like, but it's possible to solve this with vanilla Python.)