Neural Networks

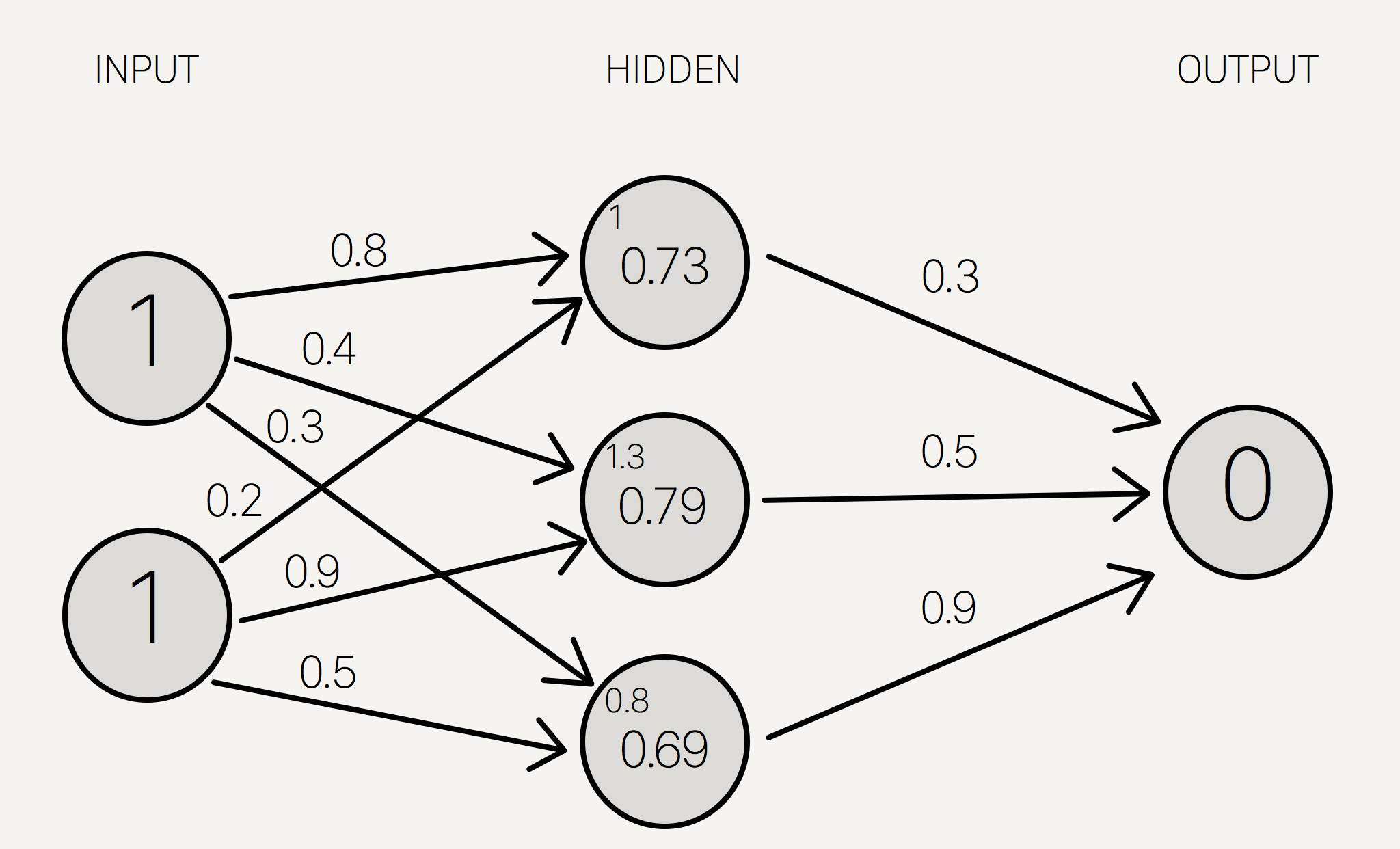
Machine Learning Mini Course

At the start of class, we looked at playground.tensorflow.org and its neural network demonstrations. We used that website to introduce the concepts of neural networks.

Additionally, if you’d like to, you may use class time to watch 3Blue1Brown’s video on neural networks. That can be found here: https://tinyurl.com/yb8fa3ze.

**Structure**

The neural network structure is as follows:



There’s an input layer, a hidden layer, and an output layer. Each circle represents a **neuron**, you can think of neurons as something that holds a number. Each arrow represents a **weight**; weights are how neurons influence each other.

The most important concept of neural networks is the idea of “hidden layers.” Basically, the **hidden layers represent intermediate concepts in accomplish your task**. You can kind of see this in the tensorflow playground, but the deeper you get into the network, the more complex your neurons can become.

**Firing A Neural Network**

Let’s start with the input layer. Given any datapoint, we simply set the input neurons to have values equal to our datapoint. For instance, here our datapoint is [1, 1].

Next, we want to calculate the values of the next few neurons. We do this by 1) taking a weighted average of the weights and previous neuron values and 2) applying an activation function.

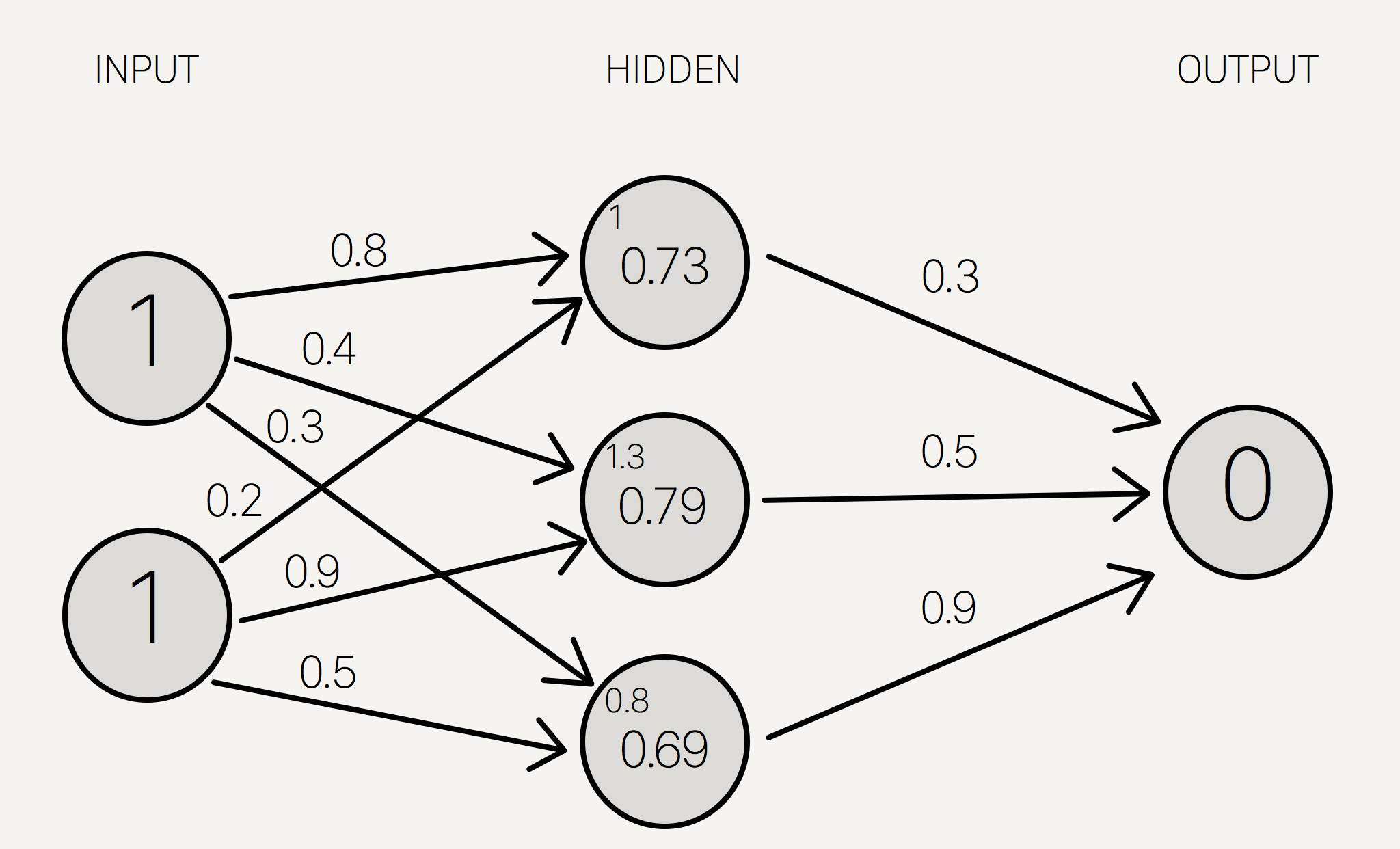
The weighted averages are shown at the top left of each neuron. For instance, for the top middle neuron, you get weighted average of 1 by doing 0.8 \* 1 + 0.2 \* 1 = 1.

Likewise, you can get the weighted average of the middle neuron to be 1.3. This is by doing 0.4 \* 1 + 0.9 \* 1 = 1.3

Then, we take the number which we found and apply an activation function, also known as a “nonlinearity”. In this example, we apply the sigmoid function to get the final value of each neuron.



Sigmoid(1) = 0.73, sigmoid(1.3) = 0.79, sigmoid(0.8) = 0.69. Now see if you can calculate the value for the last neuron. (Hint, the diagram is wrong; it shouldn’t be zero.)

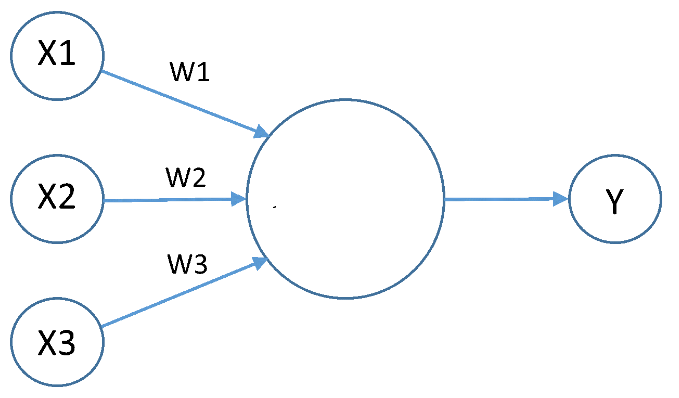


Answer: The final neuron has weighted sum of 0.3 \* 0.73 + 0.5 \* 0.79 + 0.9 \* 0.69 = 1.235

The final neuron then has value sigmoid(1.235) = 0.775

**Bias Terms**

Actually, it turns out that there’s a small detail that I forgot to mention. Let’s say we have a neuron with inputs x1, x2, x3 and weights w1, w2, w3.



Then y isn’t just sigmoid(w1x1 + w2x2 + w3x3). It’s actually sigmoid(w1x1 + w2x2 + w3x3 + b).

In essence, a neuron is like a linear model with an activation function. But to make the linear model part of the neuron sufficient, you need a bias term (b) for every single neuron.

**Motivation for Activation Functions**

And this concept of stacked linear models is very, very useful for explaining why we need activation functions.

Imagine a world which we didn’t have activation functions. Let’s say we started with inputs x1 and x2. Then we had two hidden neurons

y1 = ax1 + bx2 + c

y2 = dx1 + fx2 + g

Finally, let’s create some final neuron

z = hy1 + ky2 + m

Without any activation functions, we’d have that our neural network is just a linear model.

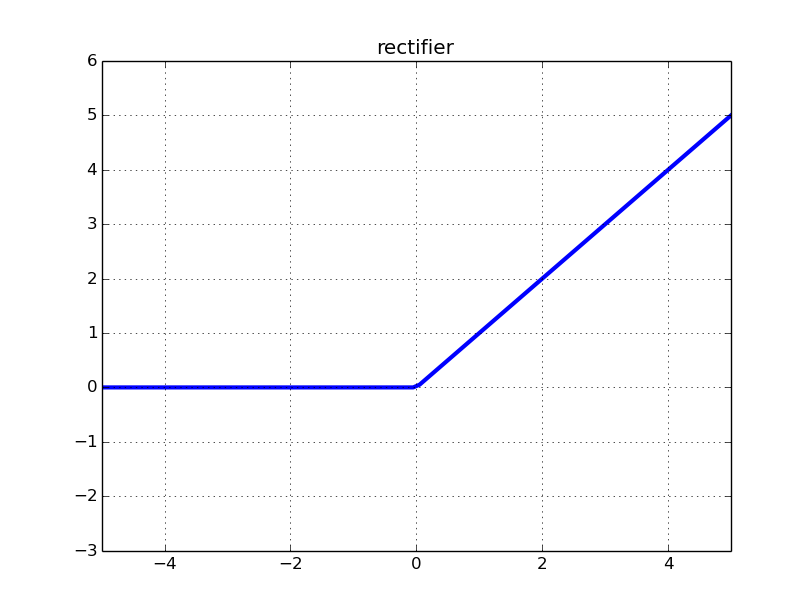
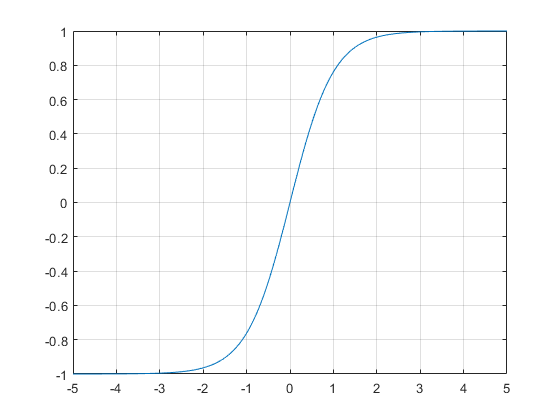
z = hy1 + ky2 + m

= h(ax1 + bx2 + c) + k(dx1 + fx2 + g) + m

= (ha + kd)x1 + (hb + kf)x2 + (hc + kg + m)

And since all ha + kd, hb + kf, and hc + kg + m are constants, this “neural network” is only a linear model.

We need activation functions (also called “nonlinearities”) to avoid being just a linear model.

Common activation functions besides sigmoid: tanh, relu

Tanh is the hyperbolic tangent (some function which maps any x into the range [-1, 1]). ReLU is literally relu(x) = max(0, x). For some reason this works really well in neural networks even if it’s really simple.