Coding the Neurons in a Neural Network

Single Neuron

Let's say we have a single neuron, and there are three inputs to this neuron. As in most cases, when you initialize parameters in neural networks, our network will have weights initialized randomly, and biases set as zero to start. Why we do this will become apparent later on. The input will be either actual training data or the outputs of neurons from the previous layer in the neural network. We're just going to make up values to start with as input for now:

```
inputs = [1, 2, 3]
```

Each input also needs a weight associated with it. Inputs are the data that we pass into the model to get desired outputs, while the weights are the parameters that we'll tune later on to get these results. Weights are one of the types of values that change inside the model during the training phase, along with biases that also change during training. The values for weights and biases are what get "trained," and they are what make a model actually work (or not work). We'll start by making up weights for now. Let's say the first input, at index 0, which is a 1, has a weight of 0.2, the second input has a weight of 0.8, and the third input has a weight of -0.5. Our input and weights lists should now be:

```
inputs = [1, 2, 3]

weights = [0.2, 0.8, -0.5]
```

Next, we need the bias. At the moment, we're modeling a single neuron with three inputs. Since we're modeling a single neuron, we only have one bias, as there's just one bias value per neuron. The bias is an additional tunable value but is not associated with any input in contrast to the weights. We'll randomly select a value of 2 as the bias for this example:

```
inputs = [1, 2, 3]

weights = [0.2, 0.8, -0.5]

bias = 2
```

This neuron sums each input multiplied by that input's weight, then adds the bias. All the neuron does is take the fractions of inputs, where these fractions (weights) are the adjustable arameters,

and adds another adjustable parameter — the bias — then outputs the result. Our output would be calculated up to this point like:

```
output = (inputs[0]*weights[0] + inputs[1]*weights[1] + inputs[2]*weights[2] + bias)

print(output)

>>> 2.3
```

All together in code, including the new input and weight, to produce output:

```
inputs = [1.0, 2.0, 3.0, 2.5]

weights = [0.2, 0.8, -0.5, 1.0]

bias = 2.0

output = (inputs[0]*weights[0] + inputs[1]*weights[1] + inputs[2]*weights[2] + inputs[3]*weights[3] + bias)

print(output)

>>> 4.8
```

A Layer of Neurons

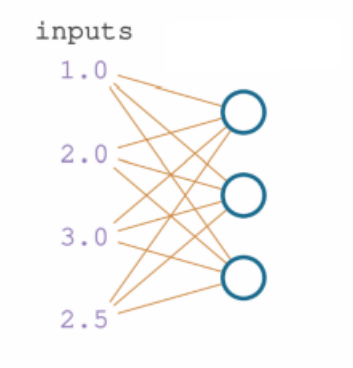
We'll keep the initial 4 inputs and set of weights for the first neuron the same as we've been using so far. We'll add 2 additional, made up, sets of weights and 2 additional biases to form 2 new neurons for a total of 3 in the layer. The layer's output is going to be a list of 3 values, not just a single value like for a single neuron.

```
inputs = [1, 2, 3, 2.5]

weights1 = [0.2, 0.8, -0.5, 1]

weights2 = [0.5, -0.91, 0.26, -0.5]
```

weights3 = [-0.26, -0.27, 0.17, 0.87]



bias1 = 2

bias2 = 3

bias3 = 0.5

outputs = [

Neuron 1:

inputs[0]*weights1[0] + inputs[1]*weights1[1] + inputs[2]*weights1[2] + inputs[3]*weights1[3] + bias1,

Neuron 2:

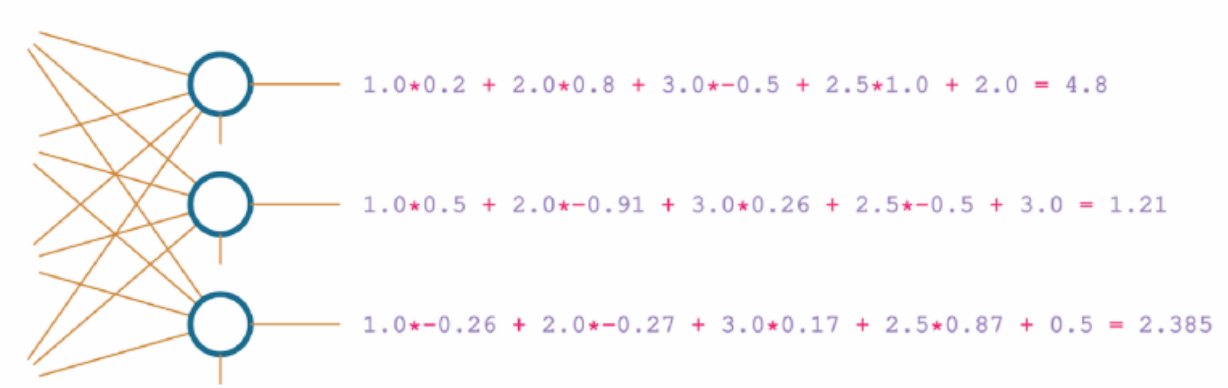
inputs[0]*weights2[0] + inputs[1]*weights2[1] + inputs[2]*weights2[2] + inputs[3]*weights2[3] + bias2,

Neuron 3:

inputs[0]*weights3[0] + inputs[1]*weights3[1] + inputs[2]*weights3[2] + inputs[3]*weights3[3] + bias3]

print(outputs)

>>> [4.8, 1.21, 2.385]



In this code, we have three sets of weights and three biases, which define three neurons. Each neuron is “connected” to the same inputs. The difference is in the separate weights and bias that each neuron applies to the input. This is called a fully connected neural network — every neuron in the current layer has connections to every neuron from the previous layer. This is a very common type of neural network, but it should be noted that there is no requirement to fully connect everything like this. At this point, we have only shown code for a single layer with very few neurons. Imagine coding many more layers and more neurons. This would get very challenging to code using our current methods. Instead, we could use a loop to scale and handle dynamically-sized inputs and layers. We’ve turned the separate weight variables into a list of weights so we can iterate over them, and we changed the code to use loops instead of the hardcoded operations.

```
inputs = [1, 2, 3, 2.5]

weights = [[0.2, 0.8, -0.5, 1], [0.5, -0.91, 0.26, -0.5], [-0.26, -0.27, 0.17, 0.87]]

biases = [2, 3, 0.5]

 # Output of current layer

layer_outputs = []

# For each neuron

for neuron_weights, neuron_bias in zip(weights, biases):

    # Zeroed output of given neuron

    neuron_output = 0

    # For each input and weight to the neuron

    for n_input, weight in zip(inputs, neuron_weights):

        # Multiply this input by associated weight

        # and add to the neuron’s output variable
```

```
        neuron_output += n_input*weight
    # Add bias
    neuron_output += neuron_bias
    # Put neuron's result to the layer's output list
    layer_outputs.append(neuron_output)
print(layer_outputs)
>>> [4.8, 1.21, 2.385]
```