

# Logits

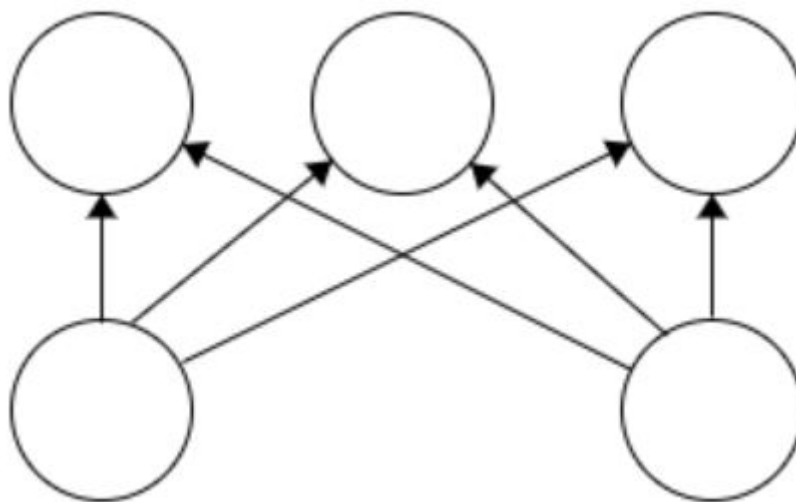
Dive into the inner layers of a neural network and understand the importance of logits.

## Chapter Goals:

- Build a single fully-connected model layer
- Output logits, AKA log-odds

### A. Fully-connected layer

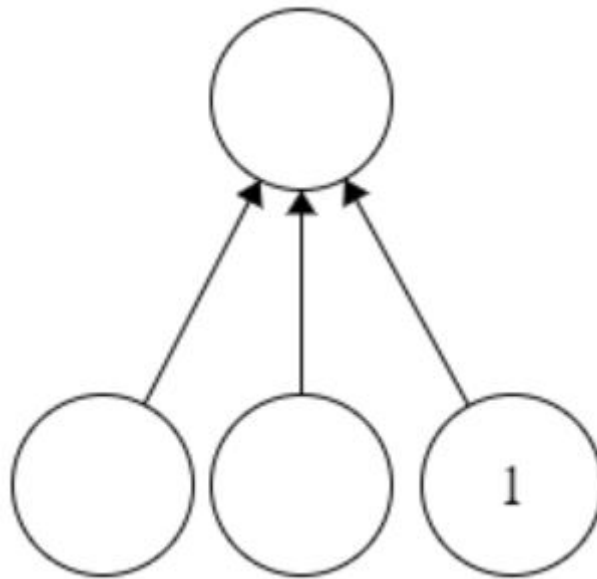
Before we can get into multilayer perceptrons, we need to start off with a single layer perceptron. The single fully-connected layer means that the input layer, i.e. `self.inputs`, is directly connected to the output layer, which has `output_size` neurons. Each of the `input_size` neurons in the input layer has a connection to each neuron in the output layer, hence the fully-connected layer.



Fully-connected layer with `input_size = 2`, `output_size = 3`

In TensorFlow, this type of fully-connected neuron layer is implemented using `tf.layers.dense` ([https://www.tensorflow.org/api\\_docs/python/tf/layers/dense](https://www.tensorflow.org/api_docs/python/tf/layers/dense)), which takes in a neuron layer and output size as required arguments, and adds a fully-connected output layer with the given size to the computation graph.

In addition to the input layer neurons, `tf.layers.dense` adds another neuron called the *bias*, which always has a value of 1 and has full connections to the output layer.

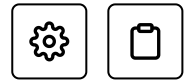
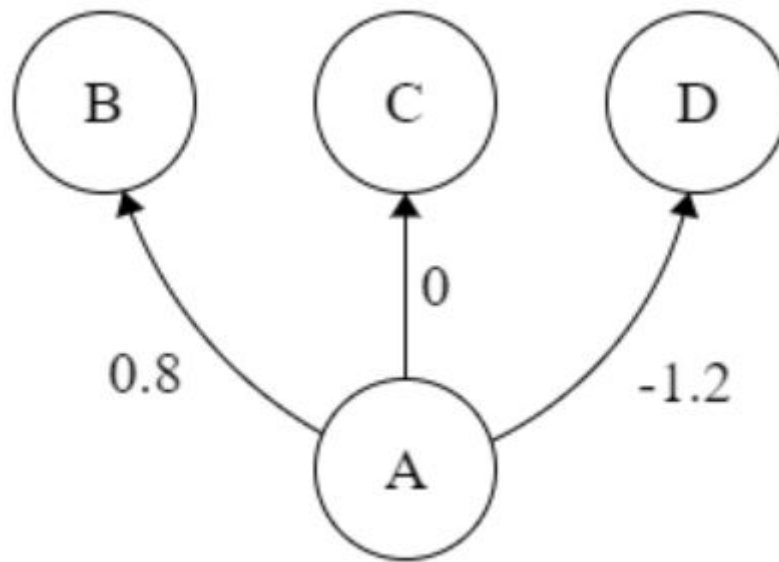


Fully-connected layer with `input_size = 2`, `output_size = 1`, and a bias neuron.

The bias neuron helps our neural network produce better results, by allowing each fully-connected layer to model a true linear combination of the input values.

## B. Weighted connections

The forces that drive a neural network are the real number weights attached to each connection. The weight on a connection from neuron **A** into neuron **B** tells how strongly **A** affects **B** as well as whether that effect is positive or negative, i.e. direct vs. inverse relationship.



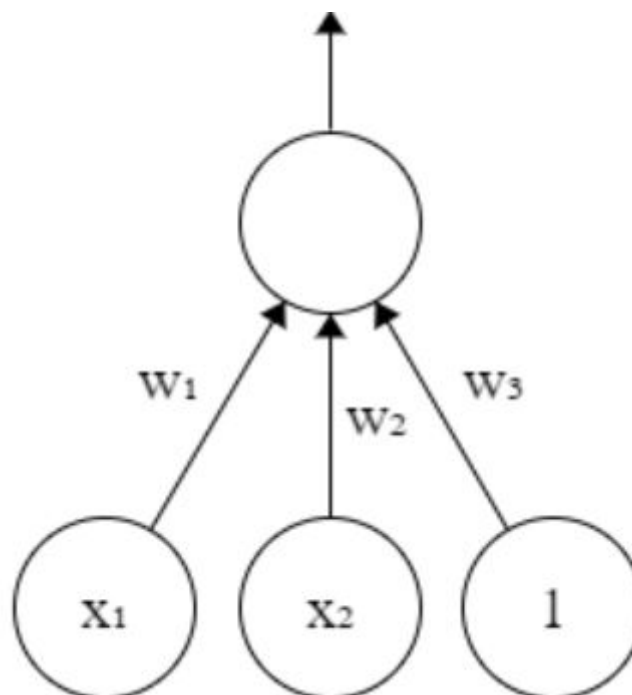
The diagram above has three weighted connections:

$A \rightarrow B$ : Direct relationship.

$A \rightarrow C$ : No relationship.

$A \rightarrow D$ : Inverse relationship.

The weighted connections allow fully-connected layers to model a linear combination of the inputs. Let's take a look at an example fully-connected layer with weights:



Fully-connected layer with input neuron values  $x_1$  and  $x_2$ , a bias neuron, and weight values  $w_1$ ,  $w_2$ , and  $w_3$



The output of this fully-connected layer is now a linear combination of the input neuron values:

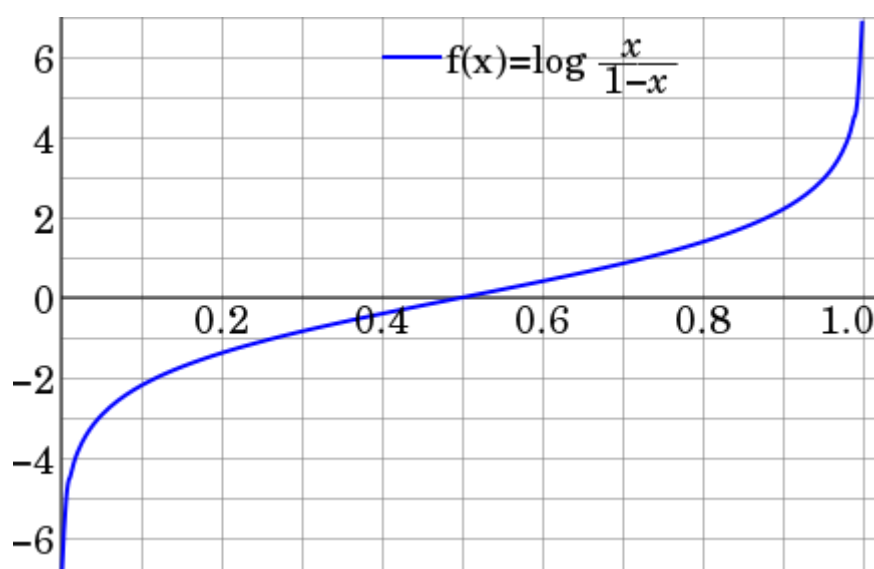
$$w_1 \cdot x_1 + w_2 \cdot x_2 + w_3$$

The logits produced by our single layer perceptron are therefore just a linear combination of the input data feature values.

Connection weights can be optimized through training (which we will cover in a later chapter), so that the logits produced by the neural network allow the model to make highly accurate predictions based on the input data.

### C. Logits

So what exactly are logits? In classification problems they represent log-odds, which maps a probability between 0 and 1 to a real number. When `output_size = 1`, our model outputs a single logit per data point. The logits will then be converted to probabilities representing how likely it is for the data point to be labeled 1 (as opposed to 0).



In the above diagram, the x-axis represents the probability and the y-axis represents the logit.

Note the vertical asymptotes at  $x = 0$  and  $x = 1$ .

We want our neural network to produce logits with large absolute values, since those represent probabilities close to 0 (meaning we are very sure the label is 0/False) or probabilities close to 1 (meaning we are very sure the label is 1/True).

## D. Regression

In the next chapter, you'll be producing actual probabilities from the logits. This makes our single layer perceptron model equivalent to logistic regression ([https://en.wikipedia.org/wiki/Logistic\\_regression](https://en.wikipedia.org/wiki/Logistic_regression)). Despite the name, logistic regression is used for classification, not regression problems. If we wanted a model for regression problems (i.e. predicting a real number such as a stock price), we would have our model directly output the logits rather than convert them to probabilities. In this case, it would be better to rename `logits`, since they don't map to a probability anymore.

## Time to Code!

The code for this chapter will build a single layer perceptron, whose output is `logits`. The code goes inside the `model_layers` function.

We're going to obtain the `logits` by applying a dense layer to `inputs` (the placeholder from Chapter 2) to return a tensor with shape `(None, self.output_size)`.

**Set `logits` equal to `tf.layers.dense` with required arguments `inputs` and `output_size`, and keyword argument `name='logits'`.**

**Then return `logits`.**

```
1 def model_layers(inputs, output_size):
2     # CODE HERE
3     pass
```



Solution



```
1 def model_layers(inputs, output_size):
```

```
1 def model_layers(inputs, output_size):  
2     logits = tf.layers.dense(inputs, output_size,  
3                               name='logits')  
4     return logits  
5  
6
```



← Back

Model Initialization

Next →

Metrics

☒ Mark as Completed

80% completed, meet the criteria and claim your course certificate!

Buy Certificate



Report an  
Issue



Ask a Question

([https://discuss.educative.io/tag/logits\\_\\_deep-learning-with-tensorflow\\_\\_machine-learning-for-software-engineers](https://discuss.educative.io/tag/logits__deep-learning-with-tensorflow__machine-learning-for-software-engineers))