Multiclass

Understand the differences between binary and multiclass classification.

Chapter Goals:

- Learn about multiclass classification
- Understand the purpose of multiple hidden layers
- Learn the pros and cons of adding hidden layers

A. Multiclass classification

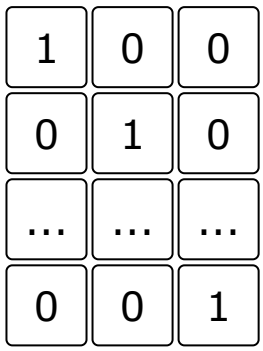
In the previous chapters we focused on binary classification, labeling whether or not an input data point has some class attribute (e.g. if it is in a circle or not). Now, we will attempt to classify input data points when there are multiple possible classes and the data point belongs to exactly one. This is referred to as multiclass classification.

The example is an extension of the previous circle example, but now there is an additional circle with radius 1 centered at the origin. The classes are now:

- **0**: Outside both circles
- **1**: Inside the smaller circle
- **2**: Outside the smaller circle but inside the larger circle

B. One-hot

Instead of representing each label as a single number, we use a *one-hot vector*. A one-hot vector has length equal to the number of classes, which is the same as `output_size`. The vector contains a single 1 at the index of the class that the data point belongs to, i.e. the *hot index*, and 0's at the other indexes. The labels now become a set of one-hot vectors for each data point:



An example set of labels. In this case there are 3 possible classes, exactly one of which is the hot index.

Another way to think about one-hot vectors is as multiple binary classification. The actual class of the data point is labeled as 1 (True), while the other classes are labeled as 0 (False).

C. Adding hidden layers

Since there are now multiple decision boundaries, it would be beneficial to either increase the size of our model's current hidden layer, `hidden1`, or add another hidden layer. Given that the decision boundaries are still relatively trivial, both methods would lead to successful models eventually. However, adding an extra hidden layer may decrease the number of training iterations needed for convergence compared to maintaining a single hidden layer.

When deciding how many hidden layers a model needs (i.e. how deep it is) and how many neurons are at each hidden layer, it is a good idea to base the decision on the problem itself. There are a few general rules of thumb, but they do not apply to every scenario. For example, it is common not to need more than 3 hidden layers in a neural network, but if you are working on a complicated problem you would most likely need more (Google's Alpha Go needed more than a dozen layers).

If you don't have much domain knowledge for the particular problem you're working on, it's usually best to only add extra layers or neurons when they're needed. The fewer layers and neurons, the faster your model trains, and the quicker you can evaluate how good it is. It then becomes easier to optimize the number of layers and neurons in your model through experimentation.

D. Overfitting

One thing to note is that the more hidden layers or neurons a neural network has, the more prone it is to overfitting the training data. Overfitting refers to the model becoming very accurate in classifying the training data, but then performing poorly on other data. Since we want models that can generalize well and accurately classify data it has never seen before, it is best to avoid going overboard in adding hidden layers.

Time to Code!

The coding exercise for this chapter involves modifying the `model_layers` function from the previous chapter. You will be adding an additional hidden layer to the model, bringing the total number of hidden layers to 2.

The additional hidden layer will go directly before the `'logits'` layer.

Set `hidden2` equal to `tf.layers.dense` with required arguments `hidden1` and `5`, as well as keyword arguments `activation=tf.nn.relu` and `name='hidden2'`.

We also need to update the layer which produces the logits, so that it takes in `hidden2` as input.

Change the first argument of `tf.layers.dense` for `logits` to be `hidden2`.

```
def model_layers(inputs, output_size):
  hidden1 = tf.layers.dense(inputs, 5,
                            activation=tf.nn.relu,
                            name='hidden1')
  logits = tf.layers.dense(hidden1, output_size,
                           name='logits')
  return logits
```