Softmax

Use the softmax function to convert a neural network from binary to multiclass classification.

Chapter Goals:

- Update the model to use the softmax function
- Perform multiclass classification

A. The softmax function

To convert the model to multiclass classification, we need to make a few changes to the metrics and training parameters. Previously, we used the sigmoid function to convert logits to probabilities, then rounded those probabilities to get a predicted label. However, now that there are multiple possible classes, we need to use the generalization of the sigmoid function, known as the softmax function.

The softmax function takes in a vector of numbers (logits for each class), and converts the numbers to a probability distribution. This means that the sum of the probabilities across all the classes equals 1, and each class's individual probability is based on how large its logit was relative to the sum of all the classes's logits.

The code below demonstrates how to use the TensorFlow `tf.nn.softmax` function to apply softmax to a tensor.

```
t = tf.constant([[0.4, -0.8, 1.3],
                 [0.2, -1.2, -0.4]])
softmax_t = tf.nn.softmax(t)
sess = tf.Session()
print('{}\n'.format(repr(sess.run(t))))
print('{}\n'.format(repr(sess.run(softmax_t))))
```

When training our model, we also replace the sigmoid cross entropy with a

softmax cross entropy, for the same reason as stated above. The cross entropy is now calculated *for each class* and then averaged at the end.
B. Predictions

Our model's prediction now becomes the class with the highest probability. Since we label each class with a unique index, we need to return the index with the maximum probability. TensorFlow provides a function that lets us do this, called `tf.argmax`.

The function takes in a required input tensor, as well as a few keyword arguments. One of the more important keyword arguments is `axis`, which tells us which dimension to retrieve the maximum index from. Setting `axis=-1` uses the final dimension, which in this case corresponds to retrieving the column index.

The code below generates multiclass predictions from probabilities using `tf.argmax`. The prediction for each row of probabilities is just the column index with the highest probability.

```
probs = tf.constant([[0.4, 0.3, 0.3],
                     [0.2, 0.7, 0.1]])
preds = tf.argmax(probs, axis=-1)
sess = tf.Session()
print('{}\n'.format(repr(sess.run(probs))))
print('{}\n'.format(repr(sess.run(preds))))
```

Time to Code!

The coding exercises for this chapter calculates multiclass predictions from logits, and then applies training with softmax cross entropy loss.

We'll first calculate the probabilities from `logits` (predefined in the backend), using the softmax function. Then we can use the `tf.argmax` function to generate the predictions.

Set `probs` equal to `tf.nn.softmax` applied to `logits`.

Set `predictions` equal to `tf.argmax` applied to `probs`, with keyword argument `axis=-1`.

```
# CODE HERE
```

Since our `labels` are one-hot vectors (predefined in the backend), we need to convert them back to class indexes to calculate our accuracy.

Set `class_labels` equal to tf.argmax applied to `labels`, with keyword argument `axis=-1`.

Set `is_correct` equal to tf.equal applied with `predictions` and `class_labels` as input arguments.

```
# CODE HERE
```

From this point, the calculation of the model's accuracy (using the `is_correct` variable is the same as in Chapter 4.

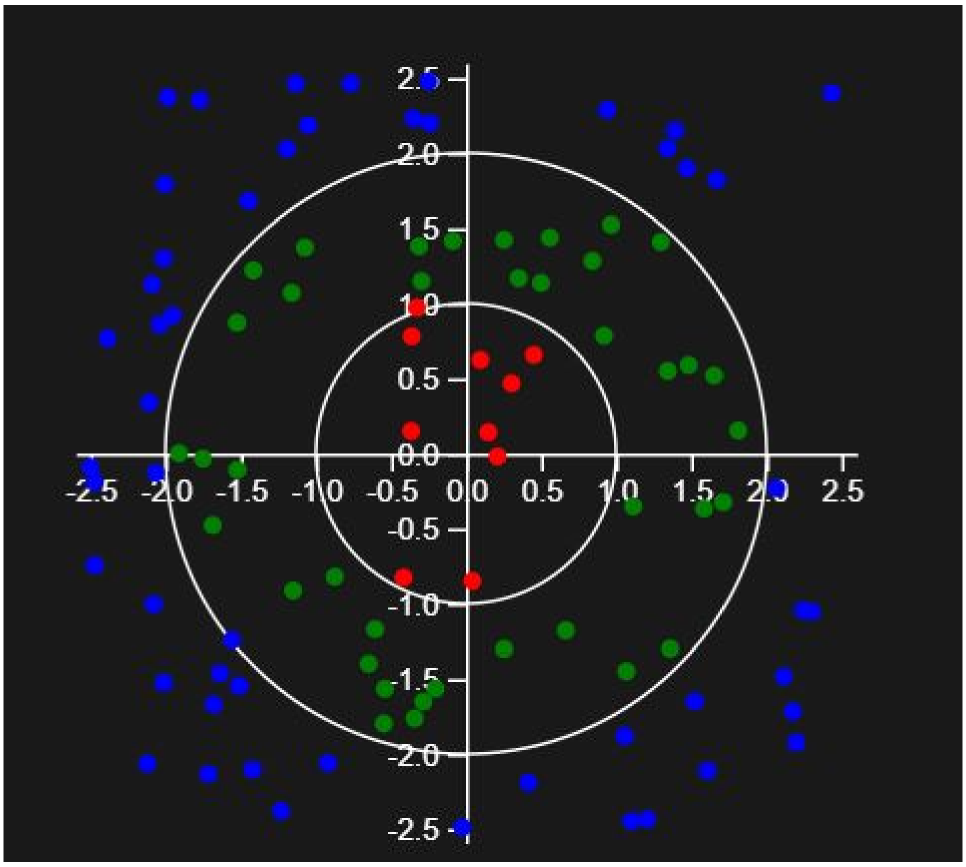
For training the model, the main change we need to make is going from sigmoid cross entropy to softmax cross entropy. In TensorFlow, softmax cross entropy is applied via the `tf.nn.softmax_cross_entropy_with_logits_v2` function.

Set `cross_entropy` equal to `tf.nn.softmax_cross_entropy_with_logits_v2`, applied with `labels` and `logits` for the `labels` and `logits` keyword arguments, respectively.

```
# CODE HERE
```

The remainder of the training optimization code is the same as in Chapter 5.

Now if we run the trained 2-hidden layer MLP model on a multiclass circle dataset, the plot will look like this:



Blue represents points that the model believes is outside both circles, green represents points the model believes is between the two circles, and red represents points the model believes is inside both circles.

As you can see, the multiple hidden layer MLP model performs quite well on this basic multiclass classification task.