Prediction Mode

Chapter Goals:

- Set up the regression function's prediction code

A. Prediction values

For the prediction mode in the regression function, we initialize and return an `EstimatorSpec` object containing a dictionary with the model's predictions. The model's predictions need to be in 2-D tensor format, with shape `(batch_size, 1)`.

Using the 1-D tensor version (which was used in calculating the loss) will result in an indexing error when making predictions on a TFRecords dataset.

```
mode = tf.estimator.ModeKeys.PREDICT
prediction_info = { 'predictions': batch_predictions }
estimator_spec = tf.estimator.EstimatorSpec(
    mode, predictions=prediction_info)
```

Creating the EstimatorSpec for predictions

Time to Code!

All code for this chapter goes in the `regression_fn` function.

The code for this chapter focuses on model predictions. The `predictions` variable used in calculating the loss is a 1-D tensor, which can cause indexing issues if we use it when running predictions on a TFRecords dataset.

Instead, we'll use `batch_predictions` (which has shape `(batch_size, 1)` in the predictions dictionary.

Outside the `if` block from the previous chapter, create another `if` block. This one should check if `mode` is equal to `tf.estimator.ModeKeys.PREDICT`.

Inside the `if` block, create a dictionary called `prediction_info`, which

Inside the `if` block, create a dictionary called `prediction_info`, which maps the string `'predictions'` to the tensor `batch_predictions`.

We can now return an `EstimatorSpec` object for the predictions.

Inside the `if` block, return `tf.estimator.EstimatorSpec` initialized with `mode` as the required argument and `prediction_info` as the `predictions` keyword argument.

```
class SalesModel(object):
  def __init__(self, hidden_layers):
    self.hidden_layers = hidden_layers

  def regression_fn(self, features, labels, mode, params):
    feature_columns = create_feature_columns()
    inputs = tf.feature_column.input_layer(features, feature_columns)
    batch_predictions = self.model_layers(inputs)
    predictions = tf.squeeze(batch_predictions)
    if labels is not None:
      loss = tf.losses.absolute_difference(labels, predictions)

    if mode == tf.estimator.ModeKeys.TRAIN:
      global_step = tf.train.get_or_create_global_step()
      adam = tf.train.AdamOptimizer()
      train_op = adam.minimize(
        loss, global_step=global_step)
      return tf.estimator.EstimatorSpec(mode, loss=loss, train_op=train_op)
    if mode == tf.estimator.ModeKeys.EVAL:
      return tf.estimator.EstimatorSpec(mode, loss=loss)
    # CODE HERE


  def model_layers(self, inputs):
    layer = inputs
    for num_nodes in self.hidden_layers:
      layer = tf.layers.dense(layer, num_nodes,
        activation=tf.nn.relu)
    batch_predictions = tf.layers.dense(layer, 1)
    return batch_predictions
```