Regression Model

Chapter Goals:

- Create the `Estimator` object for the regression model

A. `Estimator` object

The entire regression model, from training to evaluation to predictions, can be encapsulated in a single `Estimator` object. The `Estimator` object is initialized with the regression function, as well as a few keyword arguments.

One of the keyword arguments is `model_dir`, which represents the path to the directory that will contain the model's checkpoints. The checkpoints are how we save and restore the model's parameters for training, evaluation, and making predictions.

Another keyword argument we'll use is `config`, which specifies a custom configuration for the model. We can customize values like logging frequency, the maximum number of checkpoints to keep, and how often we save a summary of the training. These can all be set by creating a `RunConfig` object with the designated values.

For our regression model, the only custom configuration we'll set is the logging frequency. This refers to how frequently the model will log the loss and global step values to the screen during training. The default frequency is every 100 training steps, but since our model's training steps are very quick, we'll set the frequency to every 5000 steps.

```
config = tf.estimator.RunConfig(log_step_count_steps=5000)
regression_model = tf.estimator.Estimator(
    regression_fn,  # the model's regression function
    config=config,
    model_dir='model_ckpts')
```

Creating the Estimator object for our regression model. We set the checkpoint directory to

'model_ckpts', and set the logging frequency to every 5000 steps.

Time to Code!

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

The default configuration for training logs the loss and global step values every 100 training steps. However, since each step is very quick when training our regression model, we'll instead choose to log the values every 5000 steps.

Set `config` equal to `tf.estimator.RunConfig` initialized with `log_step_count_steps` set to `5000`.

The regression model will be an `Estimator` object, which uses `regression_fn` as the model function and `ckpt_dir` as the path to the model's checkpoints.

We'll also use the `config` variable from the previous step to ensure logging every 5000 steps during training.

Set `regression_model` equal to `tf.estimator.Estimator` initialized with `self.regression_fn` as the required argument, along with `config` and `ckpt_dir` for the `config` and `model_dir` keyword arguments.

Return `regression_model`.

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

  def create_regression_model(self, ckpt_dir):
    # CODE HERE

  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)
    if mode == tf.estimator.ModeKeys.PREDICT:
      prediction_info = {
```

```
        'predictions': batch_predictions
    }
    return tf.estimator.EstimatorSpec(mode, predictions=prediction_info)

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
```