# Training Mode

Chapter Goals:

- Set up the regression function's training code

### A. The `EstimatorSpec`

There are three phases to completing the machine learning model: training, evaluation, and prediction. With TensorFlow, we can easily bundle the three phases into a single function using `EstimatorSpec` objects for each phase.

The `EstimatorSpec` object has three modes corresponding to the three phases:

- Training: `tf.estimator.ModeKeys.TRAIN`
- Evaluation: `tf.estimator.ModeKeys.EVAL`
- Prediction: `tf.estimator.ModeKeys.PREDICT`

The `EstimatorSpec` is initialized with whatever mode corresponds to the current phase of completing the machine learning model. For training, the `EstimatorSpec` is also initialized with the model's loss and training operation (for minimizing the loss).

```
inputs = input_layer(features, cols)
```

Creating the model's input layer from a dictionary of feature values (features) and list of feature columns (cols). The tf.feature_column prefix is omitted from the input_layer function for brevity.

## Time to Code!

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

The `mode` argument for `regression_fn` lets us know whether the model is currently being run for training, evaluation, or prediction. The code for this chapter focuses on model training.

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

The rest of the code for this chapter goes inside the `if` block.

To keep track of the total number of training steps taken during multiple different training runs, we'll set up a global step object.

**Set `global_step` equal to `tf.train.get_or_create_global_step` applied with no arguments.**

We'll minimize the model's loss during training using the ADAM optimization method, via the `AdamOptimizer` object.

**Set `adam` equal to `tf.train.AdamOptimizer` initialized with no arguments.**

**Set `train_op` equal to `adam.minimize` applied with `loss` as the required argument and `global_step` as the `global_step` keyword argument.**

We can now create and return the `EstimatorSpec` object for model training.

**Return `tf.estimator.EstimatorSpec` initialized with `mode` as the required argument and `loss` and `train_op` as the `loss` and `train_op` keyword arguments.**

```
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)
    # 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
```