# Model Training

Chapter Goals:

- Train the regression model

### A. Training with the `Estimator`

Since we set up all the code for the regression model, we can now train the model using the *train.tfrecords* file we created in the Data Processing Lab. The `Estimator` object contains a `train` function that lets us easily train the model.

Interestingly, the `train` function's only required argument is a function that takes in no input arguments. This function should set up the input pipeline for the model training.

In our case, it will return the training dataset using the `create_tensorflow_dataset` function from the Data Processing Lab

```
input_fn = lambda:create_tensorflow_dataset('train.tfrecords', 50)
regression_model.train(input_fn)
```

Using the Estimator object (regression_model) to run model training. We train with a batch size of 50 and save checkpoints to the model_ckpts directory.

Since the `create_tensorflow_dataset` function repeats the dataset indefinitely for training, the above code runs training until we manually stop the process. However, if we want to run training for a fixed number of steps, we can set the `steps` keyword argument.

```
regression_model.train(input_fn, steps=20000)
```

Training for 20000 steps.

Note that the default value for `steps` is `None`, which signifies that training will run until the end of the input dataset. If the input dataset is repeated

indefinitely (as is the case in our training), setting `steps` to `None` will run training until it is manually terminated.

Code for training the regression model is shown below.

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

  def run_regression_training(self, ckpt_dir, batch_size, num_training_steps=None):
    regression_model = self.create_regression_model(ckpt_dir)
    input_fn = lambda:create_tensorflow_dataset('train.tfrecords', batch_size)
    regression_model.train(input_fn, steps=num_training_steps)

  def create_regression_model(self, ckpt_dir):
    config = tf.estimator.RunConfig(log_step_count_steps=5000)
    regression_model = tf.estimator.Estimator(
      self.regression_fn,
      config=config,
      model_dir=ckpt_dir)
    return regression_model

  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
```