Regression Function

Chapter Goals:

- Set up the regression function for the model

A. Classification vs. regression

There are two main uses of MLPs in industry: classification and regression. Classification refers to predicting a *class* for a data observation, given its feature data. Examples of classification tasks include:

- Classifying email as real or spam
- Classifying voters as Republican, Democrat, and Independent
- Classifying dolphins into a particular subspecies

The Deep Learning with Tensorflow section of Machine Learning for Software Engineers Lab goes over the details of using MLP models for classification.

The other main usage of MLPs in industry is regression. Regression refers to predicting a real number value for a data observation, given its feature data. Examples of regression tasks include:

- Predicting housing prices based on factors like region and national unemployment rate
- Predicting the weather in Celsius/Fahrenheit
- Predicting the number of points a basketball team will score in a game

For our project, we'll be creating a regression model to predict weekly sales for the departments of each store.

B. Regression loss

For regression models, there are two main loss functions that can used to train the model: mean absolute error and mean squared error. Mean absolute error takes the average absolute difference between the labels (in this case, the actual value for the weekly sales) and the model's predicted values. Mean

squared error takes a similar approach, but uses the squared difference rather than the absolute difference.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n} |label - prediction|$$

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n} (label - prediction)^2$$

Mean absolute error (MAE) vs. mean squared error (MSE) over a batch of n data observations.

The MSE (also known as L2 loss) amplifies large error values (e.g. a difference of 1000) and minimizes fractional error values (e.g. a difference of 0.01), due to the squaring operation. Since we're predicting sales values that range from about 10,000 to 40,000 dollars, the error values will be in the thousands even when the model is relatively good.

Therefore, we'll use the MAE (also known as L1 loss) to avoid unnecessary error amplification.

C. Input layer

Using the feature columns function from the Data Processing Lab, we can now create the input layer for our model. The model's input layer is just a vector that comes from combining all the numeric, indicator, and embedding feature values in the data.

The `tf.feature_column.input_layer` function allows us to easily convert a dictionary of parsed feature values and a list of feature columns into an input layer for the model.

```
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.

Using the `create_feature_columns` function from the Data Processing Lab, we'll create the input layer's feature columns.

Set `feature_columns` equal to `create_feature_columns` applied with no arguments.

Using the feature columns and feature data (i.e. the `features` input variable), we can create the model's input layer.

Set `inputs` equal to `tf.feature_column.input_layer` applied with `features` and `feature_columns` as required arguments.

Using the `model_layers` function completed in the previous chapter, we'll obtain the batch predictions for the input layer.

Set `batch_predictions` equal to `self.model_layers` applied to `inputs`.

The `batch_predictions` tensor has shape `(batch_size, 1)`, while the `labels` input argument has shape `(batch_size,)`. To calculate the model's loss, we need the two tensors to have the same shape.

We can use `tf.squeeze` on `batch_predictions` to remove the extraneous dimension of size `1`. This results in the same shape as `labels`.

Set `predictions` equal to `tf.squeeze` applied with `batch_predictions` as the only argument.

We calculate the model's loss during training and evaluation, i.e. when there are labels for the input data. The loss function will be a simple mean absolute error.

Create an `if` block that checks if `labels` is not `None`.

We'll use the `tf.losses.absolute_difference` function to calculate the mean absolute error between `labels` and `predictions`.

Inside the `if` block, set `loss` equal to `tf.losses.absolute_difference` applied with `labels` and `predictions` as required arguments.

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

  def regression_fn(self, features, labels, mode, params):
    # CODE HERE
    # 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
```