Regression

Understand the difference between regression and classification models.

Chapter Goals:

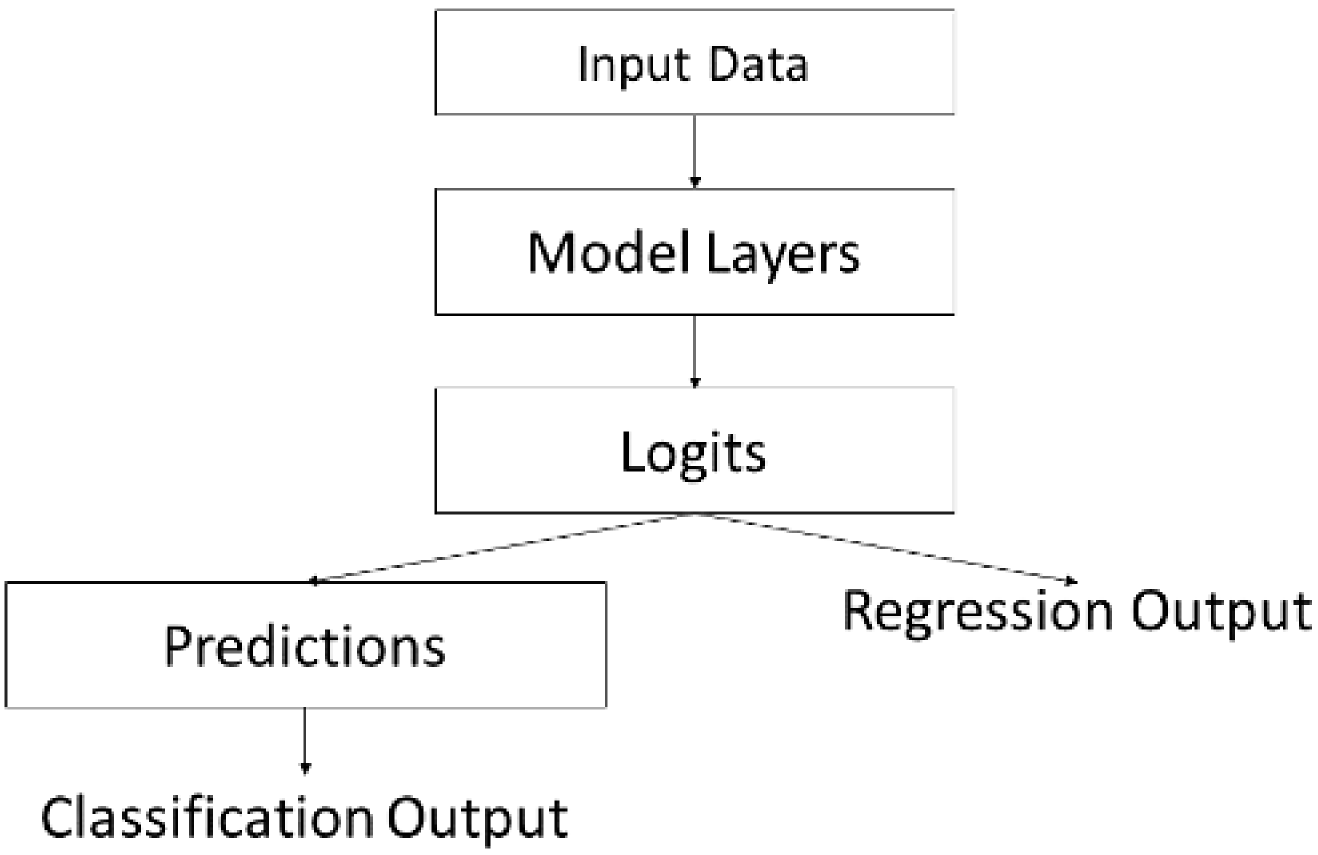
- Learn how to use a neural network for regression
- Set up the training architecture for a regression model

A. Classification vs. regression

The model we've been using so far is built for classification. This means it classifies input data into one of many different classes.

While classification is one of the most important tasks that can be done by a neural network, another common application of neural networks is *regression*. Rather than assigning the input data to a particular class, a regression model will instead output a real number value based on the input data. Regression tasks can range from predicting stock prices to predicting the future height of a child.

Classification models obtain probabilities and class predictions from the calculated logits. A regression model can just return the logits, since the logits represent the real-valued output of the model's layers.



Classification output vs. regression output.

For the remainder of this course you'll be creating the regression model that can be used for sales predictions.

B. L2 loss

When training a classification model, the loss function we use is the cross entropy loss. For regression models, we normally use the L2 loss, which is a loss function derived from the L2-norm.

In TensorFlow, the L2 loss is calculated by the `tf.nn.l2_loss` function. Note that the TensorFlow implementation omits the square root used in the L2-norm and also divides the value by 2. This makes it easier to calculate the gradient for weight optimization.

We normally use L2 loss on the subtracted difference between two scalars or tensors. The L2 loss represents how "close" the two values are.

```
import tensorflow as tf
t1 = tf.constant([9.3, 95.4, -0.3])
t2 = tf.constant([9.35, 95.4, -0.28])
t3 = tf.constant([-80.4, -95.1, 78])
with tf.Session() as sess:
  print(sess.run(tf.nn.l2_loss(t1 - t2)))
  print(sess.run(tf.nn.l2_loss(t1 - t3)))
```

An L2 loss near 0 means the two values are very close, while a large L2 loss means the values are not close at all. In our example, the elements of `t1` and `t2` are extremely close, so their L2 loss is tiny. On the other hand, `t1` and `t3` are not close at all, so their L2 loss is large.

For regression models, we calculate the L2 loss between the model's output (i.e. logits) and the labels for the input data.

Time to Code!

In this chapter you'll be completing part of the `regressor_fn` function, which is the main function for the regression model (`RegressionModel`) that will be created in this section of the course.

Specifically, you'll be completing the helper function `set_predictions_and_loss`, which sets the regression predictions and loss (if necessary).

The `logits` input argument, which represents the regression model's logits, has shape `(batch_size, 1)`. However, we want our model predictions to have the same shape as `labels`, which is `(batch_size,)`.

In this case, we use `tf.squeeze` to remove the size 1 second dimension from `logits`. The new 1-D tensor corresponds to our model predictions.

Set `self.predictions` equal to `tf.squeeze` with `logits` as the only argument.

If `labels` is specified, we want to compute the regression loss. The loss function used for regression is `tf.nn.l2_loss`.

If `labels` is not `None`, set `self.loss` equal to `tf.nn.l2_loss` applied to the difference between `labels` and `self.predictions`.

```
import numpy as np
import tensorflow as tf
```

```
class RegressionModel(object):
    def __init__(self, output_size):
        self.output_size = output_size

    # Helper for regressor_fn
    def set_predictions_and_loss(self, logits, labels):
        # CODE HERE
        pass

    # The function for the regression model
    def regressor_fn(self, features, labels, mode, params):
        inputs = tf.feature_column.input_layer(features, params['feature_columns'])
        layer = inputs
        for num_nodes in params['hidden_layers']:
            layer = tf.layers.dense(layer, num_nodes,
                activation=tf.nn.relu)
        logits = tf.layers.dense(layer, self.output_size,
            name='logits')
        self.set_predictions_and_loss(logits, labels)
        if mode == tf.estimator.ModeKeys.TRAIN:
            self.global_step = tf.train.get_or_create_global_step()
            adam = tf.train.AdamOptimizer()
            self.train_op = adam.minimize(
                self.loss, global_step=self.global_step)
        if mode == tf.estimator.ModeKeys.EVAL:
            pass
        if mode == tf.estimator.ModeKeys.PREDICT:
            pass
```