Model Initialization

Learn about the input and output layers of a neural network.

Chapter Goals:

- Define a class for an MLP model
- Initialize the model

A. Placeholder

When building the computation graph of our model, `tf.placeholder` acts as a "placeholder" for the input data and labels. Without the `tf.placeholder`, we would not be able to train our model on real input data.

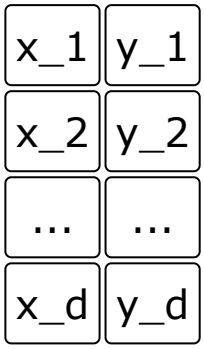
A `tf.placeholder` takes in a required first argument and two keyword arguments called `shape` and `name`.

The required argument for the placeholder is its type. In **Deep Learning with TensorFlow**, our input data is pairs of (x, y) points, so `self.inputs` has type `tf.float32`. The labels (which are explained in a later chapter) have type `tf.int32`.

The `name` keyword argument allows us to give a custom name to our placeholder, making it easier for debugging.

B. Shapes and dimensions

The `shape` argument is a tuple of integers representing the size of each of the placeholder tensor's dimensions. In **Deep Learning with TensorFlow**, and many real world problems, the shape of the input data will be a two integer tuple. If we view the input data as coming from a data table, the `shape` is akin to the dimensions of the table.



The shape of this data is (d, 2). There are d data points, each of dimension 2.

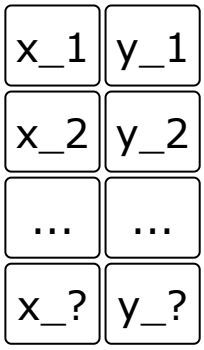
The first integer represents the number of data points we pass in (i.e. number of rows in the data table). When training the model, we refer to the first dimension as the *batch size*.

The second integer represents the number of features in the dataset (i.e. number of columns). In the remaining chapters of **Deep Learning with TensorFlow**, our input data will have two features: the *x* and *y* coordinates of the data point. So `input_size` is 2.

Each data point also has a label, which is used to identify and categorize the data. The labels we use for our model's data will have a two dimension shape, with `output_size` as the second dimension (the first dimension is still the batch size). The `output_size` refers to the number of possible *classes* a label can have (explained in a later chapter).

C. Different amounts of data

One thing to note about TensorFlow is the use of `None` in place of a dimension size. When we use `None` in the `shape` tuple, we are allowing that dimension to take on any size.



The shape of this data is (None, 2). There can be variable number of data points, each of dimension 2.

This is particularly useful because we will use our neural network on input data with different input sizes.

When we input multiple input data for our neural network, we don't actually use multiple neural networks. Rather, we use the same neural network on each of the points simultaneously to obtain the network's output for each point.

Time to Code!

The code for this chapter will focus on initializing the placeholders for the MLP model.

First you'll define the placeholder for the model's input data, in the function `init_inputs`. Note that TensorFlow is already imported in the backend via the import statement `import tensorflow as tf`.

Set `inputs` equal to a `tf.placeholder` with data type `tf.float32` and keyword arguments `shape=(None, input_size)` and `name='inputs'`.

Then return `inputs`.

```
def init_inputs(input_size):
  # CODE HERE
  pass
```

Next, we'll define placeholders for the labels, in the function `init_labels`.

Set `labels` equal to a `tf.placeholder` with data type `tf.int32` and keyword arguments `shape=(None, output_size)` and `name='labels'`.

Then return `labels`.

```
def init_labels(output_size):
  # CODE HERE
  pass
```