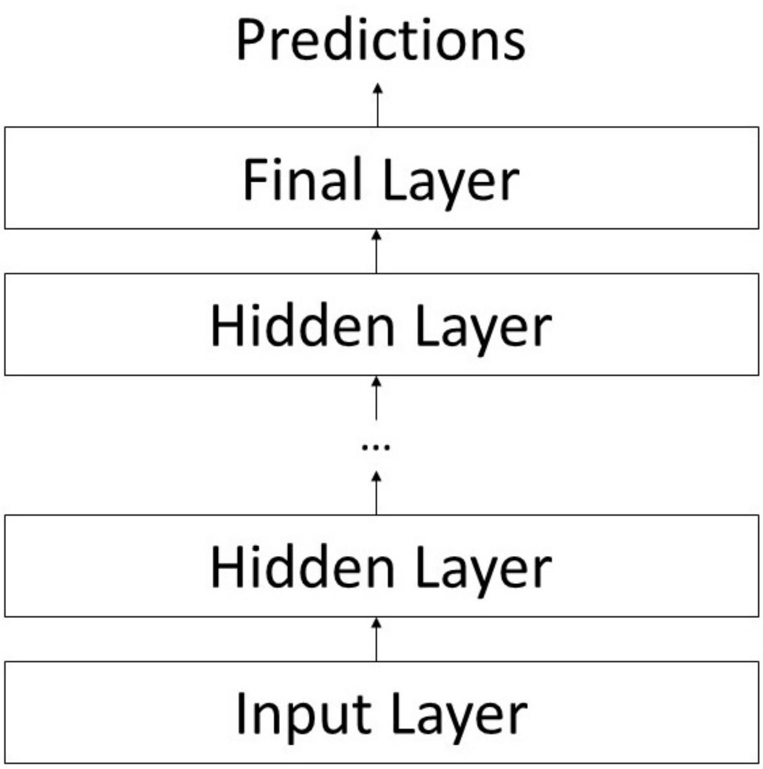
Model Layers

Chapter Goals:

- Create the generalizable function for the MLP model's layers

A. MLP architecture

Our project's model follows the standard MLP architecture. This means that it is made up of multiple fully-connected layers, where each hidden layer uses ReLU activation and the final layer uses no activation. The input layer for the MLP consists of a batch of data observations from the input pipeline (more on this later).



The architecture of the MLP model used for sales predictions.

Larger models (i.e. more hidden layers and nodes) have higher potential to make more accurate predictions, but they can also take longer to train and have a higher chance of overfitting. It's good to experiment with different have a higher chance of overfitting. It's good to experiment with different model sizes, so we can ultimately choose the best model. This is why we use

an evaluation set in addition to the training set, to compare different model configurations and see which configuration performs best on new data.

For our MLP model, we'll start off with 2 hidden layers. The first hidden layer contains 200 nodes, while the second contains 100. This equates to the list `[200, 100]` for initializing the `SalesModel` class object.

Time to Code!

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

The first layer for the MLP is the input layer. This corresponds to the `inputs` argument of the function.

Set `layer` equal to `inputs`.

The `SalesModel` class is initialized with a `hidden_layers` argument. This is a list of integers, where the integer at index *i* represents the number of nodes in hidden layer *i* of the MLP.

Create a `for` loop that iterates through `self.hidden_layers` using a variable called `num_nodes`.

Each hidden layer of the MLP is a fully-connected layer with ReLU activation, using the previous layer's output as the input.

Inside the `for` loop, set `layer` equal to `tf.layers.dense` applied with `layer` and `num_nodes` as required arguments, along with `tf.nn.relu` as the `activation` keyword argument.

The model's predictions for the input batch of data observations is the output from the MLP's final layer. The final layer has one node, since sales predictions are a single number, and doesn't use any activation function.

Outside the `for` loop, set `batch_predictions` equal to `tf.layers.dense` applied with `layer` and `1` as required arguments.

Return `batch_predictions`.

```
class SalesModel(object):
```

```
  def __init__(self, hidden_layers):
    self.hidden_layers = hidden_layers


  def model_layers(self, inputs):
    # CODE HERE
```