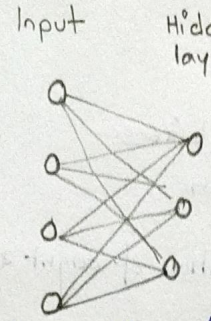
03.1 CHAPTER 3 - Adding layers

- Till now, we build only one layer. Neural network can be deep when they have 2 or more hidden layers.
- A hidden layer isn't an input or output layer, but layers inbetween these endpoints have value that we don't necessarily deal with, hence the name "hidden".

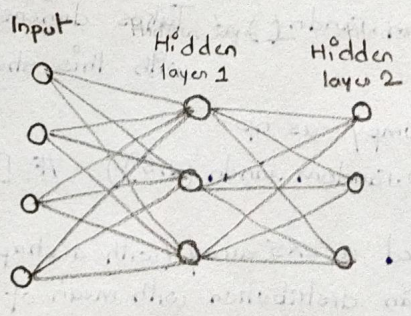


- Here we have 4 features, to our first hidden layer which we can see 3 set of weights, with 4 value each
- Since we have 3 weight sets, we have 3 neurons in first hidden layer.
- Each neuron has a unique set of weights of which we have 4 (as their are 4 inputs) so shape is (3,4)

Input layer with 4 features
into a hidden layer with 3 neuron
Now we wish to another layer. For now let's create two weights and biases

```
import numpy as np
inputs = [[1.2, 3.2.5], [2.0, 5.0, -1.0, 2], [-1.5, 2.7, 3.3, -0.8]]
weights = [[0.2, 0.8, -0.5, 1], [0.5, -0.91, 0.26, -0.5], [-0.26, -0.27, 0.17, 0.87]]
biases = [2, 3, 0.5]
weights2 = [[0.1, -0.14, 0.5], [-0.5, 0.12, -0.33], [-0.44, 0.73, -0.13]]
biases2 = [-1, 2, -0.5]
layer1_output = np.dot(inputs, np.array(weights).T) + biases
layer2_output = np.dot(layer1_output, np.array(weights2).T) + biases2
print(layer2_output)  # array [[0.50  -1.04  -2.03],
                      #        [0.24  -2.73  -5.76],
                      #        [-0.99  1.41  -0.35]])
```

Our neural networks will look something like.
4 feature input into 2 hidden layers of 3 neuron each



Dense Layer Class

- Normally, we have huge dataset so we cannot hand type our data, we should create similar for our various types of neural network

So for a Dense input or fully connected layer, will begin with two methods

```
class Layer_Dense:

    def __init__(self, n_inputs, n_neurons):
        #initialize weight and biases
        pass    # using pass statement as a placeholder.

    #Forward pass
    def forward(self, inputs):
        # calculate output values from inputs, weight and biases
        pass    # using pass statement as a placeholder.
```

Let build a sample Dense layer class, add random initialization of weight & biases

```
    def __init__(self, n_inputs, n_neurons):
        self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)
        Self.biases = np.zero((1, n_neurons))
```

Here, we're setting weights to be random and biases to be 0.
Note that we're initializing weights to be (input, neurons) rather than (neuron, inputs)

Why zero bias?
- In specific scenario, many samples containing values of 0, bias ensure atleast neuron start calculating initially
- So most common initialization is 0

np.random.randn() - produces a Gaussian distribution with mean of 0 and variance 1.
- Generate random number both positive, negative centered at 0 mean value close to 0.

We are going to multiply this Gaussian distribution by weight 0.01 to generate small magnitude. Otherwise, model will take lot of time to training. Idea is to start model with non-zero value small enough that they won't affect training.

np.random.randn() - Takes dimension sizes as parameters and create output array with this shape.

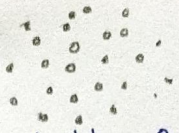
```
import numpy as np
print(np.random.randn(2,5))  # [[ 1.76  0.40  0.97  2.24  1.86
                                 -0.97  0.95 -0.15 -0.10  0.41]]
```

It returned a 2x5 array (with a shape (2,5)) with data random sample from a Gaussian distribution with mean of 0.

```
print(np.zeroes(2,5))  # [[0 0 0 0 0
                           0 0 0 0 0]]
```

We initialize the biases with the shapes of (1, n_neurons) as row vectors, which will let us add it to result of dot product later, without additional operation like transposition.

03.3 Full code upto this point :

Sample spiral data 

```
import numpy as np
import nnfs
from nnfs.datasets import spiral_data
nnfs.init()  # will set state=0, sample will be same.

class Layer_Dense:
    # layer initialization
    def __init__(self, n_inputs, n_neurons):
        # initialize weight and biases
        self.weights = 0.01 * np.random.randn(n_inputs, n_neurons)
        self.biases = np.zeros((1, n_neurons))

    # Forward pass
    def forward(self, inputs):
        # calculate output values from input, weight and biases
        self.output = np.dot(inputs, self.weights) + self.biases

# Create dataset, spiral dataset inbuilt.
X, y = spiral_data(samples=100, classes=3)
# Create Dense layer with 2 input feature and 3 output values.
dense1 = Layer_Dense(2, 3)
# Perform a forward pass of our training data through this layer.
dense1.forward(X)
# Let see first few samples of output.
print(dense1.output[:5])   # [[ 0.00   0.00   0.00],
                           #  [-1.04   1.39  -4.70],
                           #  [ 1.29   1.31  -8.01],
                           #  [ 1.79   3.59  -5.02],
                           #  [ 5.69   5.68  -8.79]]
```

] nnfs has inbuild spiral datasets which we can have target column as per our need.

Output we have 5 rows, 3 values each. Each of those 3 values is the value from the 3 neurons in dense1 layer. after passing in each of the sample.