Part 1: Theoretical Understanding of FFNN

A **Feedforward Neural Network (FFNN)** is a type of artificial neural network where data flows only in one direction—from input nodes, through hidden nodes (if any), to the output nodes. In an FFNN, each neuron computes a weighted sum of its inputs, adds a bias, and applies an activation function to produce an output.

Components of the FFNN

1. **Input Layer**:
   - This layer represents the input features. In our example, we have three input features:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} \text{Hours of study} \\ \text{Attendance percentage} \\ \text{Practice problems solved} \end{bmatrix}$$

2. **Hidden Layer**:
   - The hidden layer consists of neurons that apply a weighted sum of inputs and a bias, followed by an activation function (ReLU in this case).
   - For each hidden neuron h_i, we can compute:

$$a_i = \sum_{j=1}^{n} w_{ij} x_j + b_i$$

     where:

     - w_{ij} is the weight between input x_j and hidden neuron h_i,
     - b_i is the bias of hidden neuron h_i,
     - x_j are the input features.

3. **Output Layer**:
   - The output layer neuron computes a weighted sum of the hidden layer outputs and adds a bias. For binary classification, we use a **sigmoid activation function** to produce a probability.
   - The output neuron's value, y, can be computed as:

$$y = \sigma\left(\sum_{i=1}^{m} w_{\text{out},i} h_i + b_{\text{out}}\right)$$

     where:

     - $w_{\text{out},i}$ is the weight from hidden neuron h_i to the output neuron,
     - b_{out} is the bias of the output neuron,

- $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Part 2: Mathematical Formulation Using Linear Algebra

To make calculations more efficient, we can represent inputs, weights, and biases using vectors and matrices.

Hidden Layer Calculation

1. **Input Feature Vector $\mathbf{x}$**:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 5 \\ 80 \\ 20 \end{bmatrix}$$

2. **Hidden Layer Weights Matrix $\mathbf{W}_{\text{hidden}}$**:

$$\mathbf{W}_{\text{hidden}} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{bmatrix} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.7 & 0.3 \end{bmatrix}$$

3. **Hidden Layer Bias Vector $\mathbf{b}_{\text{hidden}}$**:

$$\mathbf{b}_{\text{hidden}} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

4. **Weighted Sum for Hidden Layer**: The output of the hidden layer, before activation, is:

$$\mathbf{a} = \mathbf{W}_{\text{hidden}} \cdot \mathbf{x} + \mathbf{b}_{\text{hidden}}$$

   Substituting values:

$$\mathbf{a} = \begin{bmatrix} 0.5 & 0.3 & 0.2 \\ 0.4 & 0.7 & 0.3 \end{bmatrix} \begin{bmatrix} 5 \\ 80 \\ 20 \end{bmatrix} + \begin{bmatrix} 0.1 \\ 0.2 \end{bmatrix}$$

5. **Activation Function (ReLU)**: After applying the ReLU activation function, we get the hidden layer output $\mathbf{h}$:

$$\mathbf{h} = \text{ReLU}(\mathbf{a})$$

Output Layer Calculation

1. **Output Layer Weights Vector $\mathbf{W}_{\text{out}}$**:

$$\mathbf{W}_{\text{out}} = \begin{bmatrix} w_{31} & w_{32} \end{bmatrix} = \begin{bmatrix} 0.6 & 0.8 \end{bmatrix}$$

2. **Output Layer Bias b_{out}**:

$$b_{\text{out}} = 0.1$$

3. **Weighted Sum for Output Layer**: The output neuron's pre-activation value a_{out} is:

$$a_{\text{out}} = \mathbf{W}_{\text{out}} \cdot \mathbf{h} + b_{\text{out}}$$

4. **Activation Function (Sigmoid)**: Applying the sigmoid activation function gives us the final output y:

$$y = \sigma(a_{\text{out}}) = \frac{1}{1 + e^{-a_{\text{out}}}}$$

Part 3: Example Calculation (Forward Propagation)

Let's go through an example using the specified values for the input features, weights, and biases.

Step 1: Calculate Hidden Layer Outputs

1. **Calculate Weighted Sum for Hidden Neurons**:
   - For Neuron 1:
   $$a_1 = (0.5 \cdot 5) + (0.3 \cdot 80) + (0.2 \cdot 20) + 0.1 = 28.6$$
   - For Neuron 2:
   $$a_2 = (0.4 \cdot 5) + (0.7 \cdot 80) + (0.3 \cdot 20) + 0.2 = 64.2$$

   So, $\mathbf{a} = \begin{bmatrix} 28.6 \\ 64.2 \end{bmatrix}$.

2. **Apply ReLU Activation Function**:

$$\mathbf{h} = \text{ReLU}(\mathbf{a}) = \begin{bmatrix} \max(0, 28.6) \\ \max(0, 64.2) \end{bmatrix} = \begin{bmatrix} 28.6 \\ 64.2 \end{bmatrix}$$

Step 2: Calculate Output Layer

1. **Calculate Weighted Sum for Output Neuron**:

$$a_{\text{out}} = (0.6 \cdot 28.6) + (0.8 \cdot 64.2) + 0.1 = 68.62$$

2. **Apply Sigmoid Activation Function**:

$$y = \sigma(a_{\text{out}}) = \frac{1}{1 + e^{-68.62}} \approx 1.0$$

Final Prediction

Since $y \approx 1.0$, the network predicts that the student will **Pass** the exam.

Key Points of Forward Pass:

1. **Fixed Weights and Biases**: In forward propagation, the network uses pre-set weights and biases to compute outputs. No learning or adjustment of weights occurs here; learning happens during backpropagation.
2. **Layer-by-Layer Calculation**: The forward pass proceeds layer by layer, computing weighted sums and applying activation functions.
3. **Non-Linearity through Activation Functions**: Activation functions, like ReLU and sigmoid, introduce non-linearity, allowing the network to capture more complex relationships in the data.
4. **Probability Output**: In binary classification, the sigmoid function in the output layer maps the result to a probability between 0 and 1, making it easy to interpret predictions.
5. **Efficient Computation with Linear Algebra**: Representing inputs, weights, and biases as vectors and matrices enables efficient computation, especially for larger networks.