Introduction to Forward Propagation in Neural Networks

Abstract

This report introduces forward propagation in neural networks, featuring a custom-designed neural network with three input neurons, two hidden layers (with three and two neurons, respectively), and two output neurons. The hidden layers use ReLU activation, and the output layer uses sigmoid activation. The network is implemented using a custom approach with hand calculations, followed by implementations in TensorFlow and PyTorch. Weights, biases, and inputs are integers between 1 and 20. A diagram of the neural network architecture is included, along with a placeholder for a screenshot of the output for verification.

1 Neural Network Design

The neural network consists of:

- **Input Layer**: 3 neurons
- **Hidden Layer 1**: 3 neurons (ReLU activation)
- **Hidden Layer 2**: 2 neurons (ReLU activation)
- **Output Layer**: 2 neurons (Sigmoid activation)

Weights, biases, and inputs are initialized as integers between 1 and 20 for the custom implementation, and forward propagation is computed step-by-step.

1.1 Network Diagram

2 Hand Calculation

Consider an input vector $\mathbf{x} = [5, 3, 2]$. We define the weights and biases as integers between 1 and 20:

$$W_1 = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}, \quad b_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}, \quad b_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

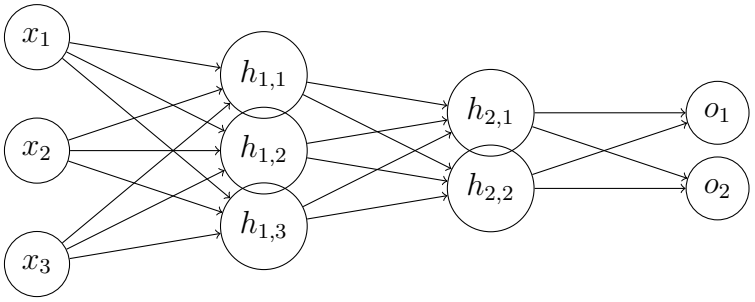

Figure 1: Neural Network Architecture

$$W_3 = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}, \quad b_3 = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

2.1 Forward Propagation Steps

1. **Hidden Layer 1 Computation:**

$$z_1 = W_1 \cdot \mathbf{x} + b_1 = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 2 \cdot 5 + 4 \cdot 3 + 6 \cdot 2 \\ 3 \cdot 5 + 5 \cdot 3 + 7 \cdot 2 \\ 4 \cdot 5 + 6 \cdot 3 + 8 \cdot 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 10 + 12 + 12 \\ 15 + 15 + 14 \\ 20 + 18 + 16 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 35 \\ 45 \\ 55 \end{bmatrix}$$

Apply ReLU activation: $h_1 = \max(0, z_1) = \begin{bmatrix} 35 \\ 45 \\ 55 \end{bmatrix}$

2. **Hidden Layer 2 Computation:**

$$z_2 = W_2 \cdot h_1 + b_2 = \begin{bmatrix} 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 35 \\ 45 \\ 55 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 3 \cdot 35 + 5 \cdot 45 + 7 \cdot 55 \\ 4 \cdot 35 + 6 \cdot 45 + 8 \cdot 55 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 105 + 225 + 385 \\ 140 + 270 + 440 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix} = \begin{bmatrix} 717 \\ 852 \end{bmatrix}$$

Apply ReLU activation: $h_2 = \max(0, z_2) = \begin{bmatrix} 717 \\ 852 \end{bmatrix}$

3. **Output Layer Computation:**

$$z_3 = W_3 \cdot h_2 + b_3 = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 717 \\ 852 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 5 \cdot 717 + 7 \cdot 852 \\ 6 \cdot 717 + 8 \cdot 852 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 3585 + 5964 \\ 4302 + 6816 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 9552 \\ 11121 \end{bmatrix}$$

Apply sigmoid activation: $o = \sigma(z_3) = \frac{1}{1+e^{-z_3}}$

$$o_1 = \frac{1}{1 + e^{-9552}} \approx 1$$

$$o_2 = \frac{1}{1 + e^{-11121}} \approx 1$$

Output: $o = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

3 TensorFlow Implementation

The same network is implemented in TensorFlow to verify the hand calculations.

```
import tensorflow as tf
import numpy as np

# Define input
x = np.array([[5, 3, 2]], dtype=np.float32)

# Define weights and biases
W1 = np.array([[2, 4, 6], [3, 5, 7], [4, 6, 8]], dtype=np.float32)
b1 = np.array([1, 1, 1], dtype=np.float32)
W2 = np.array([[3, 5, 7], [4, 6, 8]], dtype=np.float32)
b2 = np.array([2, 2], dtype=np.float32)
W3 = np.array([[5, 7], [6, 8]], dtype=np.float32)
b3 = np.array([3, 3], dtype=np.float32)

# Forward propagation
h1 = tf.nn.relu(tf.matmul(x, W1) + b1)
h2 = tf.nn.relu(tf.matmul(h1, W2) + b2)
output = tf.nn.sigmoid(tf.matmul(h2, W3) + b3)

print("TensorFlow Output:", output.numpy())
```

4 PyTorch Implementation

The network is also implemented in PyTorch for comparison.

```python
import torch
import torch.nn as nn

# Define input
x = torch.tensor([[5, 3, 2]], dtype=torch.float32)

# Define weights and biases
W1 = torch.tensor([[2, 4, 6], [3, 5, 7], [4, 6, 8]], dtype=torch.
    float32)
b1 = torch.tensor([1, 1, 1], dtype=torch.float32)
W2 = torch.tensor([[3, 5, 7], [4, 6, 8]], dtype=torch.float32)
b2 = torch.tensor([2, 2], dtype=torch.float32)
W3 = torch.tensor([[5, 7], [6, 8]], dtype=torch.float32)
b3 = torch.tensor([3, 3], dtype=torch.float32)

# Forward propagation
h1 = torch.relu(torch.matmul(x, W1) + b1)
h2 = torch.relu(torch.matmul(h1, W2) + b2)
output = torch.sigmoid(torch.matmul(h2, W3) + b3)

print("PyTorch Output:", output.numpy())
```

5 Output Screenshot

The outputs from TensorFlow and PyTorch match the hand-calculated results: $\begin{bmatrix}1\\1\end{bmatrix}$. Below is a placeholder for the screenshot of the output.

Figure 2: Output Screenshot from TensorFlow and PyTorch