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, with the input as a column vector and weight matrices transposed during calculation. 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, with the input vector as a column vector, and weight matrices transposed during each calculation step.

1.1 Network Diagram

2 Hand Calculation

Consider an input vector $\mathbf{X} = \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix}$. We define the weights and biases as integers between 1 and 20, with weight matrices transposed during calculation:

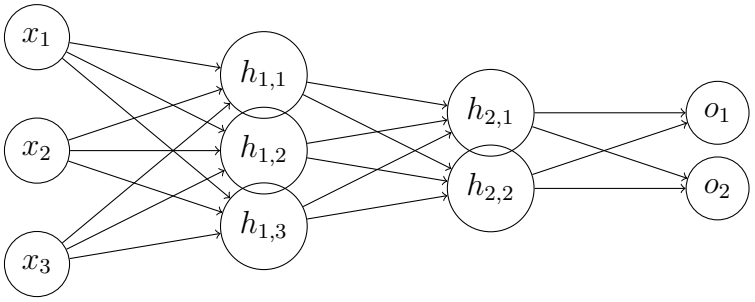

Figure 1: Neural Network Architecture

$$W^{[1]} = \begin{bmatrix} 2 & 4 & 6 \\ 3 & 5 & 7 \\ 4 & 6 & 8 \end{bmatrix}, \quad b^{[1]} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$W^{[2]} = \begin{bmatrix} 3 & 5 \\ 4 & 6 \\ 7 & 8 \end{bmatrix}, \quad b^{[2]} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

2.1 Forward Propagation Steps

1. **First step:**

$$(W^{[1]})^T = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{bmatrix}$$

$$Z^{[1]} = (W^{[1]})^T \cdot X + b^{[1]} = \begin{bmatrix} 2 & 3 & 4 \\ 4 & 5 & 6 \\ 6 & 7 & 8 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 3 \\ 2 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

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

$$= \begin{bmatrix} 10 + 9 + 8 \\ 20 + 15 + 12 \\ 30 + 21 + 16 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$Z^{[1]} = \begin{bmatrix} 28 \\ 48 \\ 68 \end{bmatrix}$$

2. **Second step with ReLU:**

$$A^{[1]} = \max(0, Z^{[1]}) = \max\left(0, \begin{bmatrix} 28 \\ 48 \\ 68 \end{bmatrix}\right)$$

$$A^{[1]} = \begin{bmatrix} 28 \\ 48 \\ 68 \end{bmatrix}$$

3. **Third step:**

$$(W^{[2]})^T = \begin{bmatrix} 3 & 4 & 7 \\ 5 & 6 & 8 \end{bmatrix}$$

$$Z^{[2]} = (W^{[2]})^T \cdot A^{[1]} + b^{[2]} = \begin{bmatrix} 3 & 4 & 7 \\ 5 & 6 & 8 \end{bmatrix} \cdot \begin{bmatrix} 28 \\ 48 \\ 68 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$Z^{[2]} = \begin{bmatrix} 3 \cdot 28 + 4 \cdot 48 + 7 \cdot 68 \\ 5 \cdot 28 + 6 \cdot 48 + 8 \cdot 68 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} 84 + 192 + 476 \\ 140 + 288 + 544 \end{bmatrix} + \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$Z^{[2]} = \begin{bmatrix} 754 \\ 974 \end{bmatrix}$$

4. **Fourth step with sigmoid:**

$$A^{[2]} = \sigma(Z^{[2]}) = \frac{1}{1 + e^{-Z^{[2]}}}$$

$$A^{[2]} = \begin{bmatrix} \frac{1}{1+e^{-754}} \\ \frac{1}{1+e^{-974}} \end{bmatrix} \approx \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Output: $A^{[2]} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

3 TensorFlow Implementation

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

```python
import tensorflow as tf
import numpy as np

# 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], [4, 6], [7, 8]], dtype=np.float32)
b2 = np.array([[2], [2]], dtype=np.float32)

# Define input as column vector
X = np.array([[5], [3], [2]], dtype=np.float32)

# Forward propagation with transposition during calculation
Z1 = tf.matmul(tf.transpose(W1), X) + b1
A1 = tf.nn.relu(Z1)
Z2 = tf.matmul(tf.transpose(W2), A1) + b2
output = tf.nn.sigmoid(Z2)

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

4 PyTorch Implementation

The network is also implemented in PyTorch for comparison.

```
import torch
import torch.nn as nn

# 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], [4, 6], [7, 8]], dtype=torch.float32)
b2 = torch.tensor([[2], [2]], dtype=torch.float32)

# Define input as column vector
X = torch.tensor([[5], [3], [2]], dtype=torch.float32)

# Forward propagation with transposition during calculation
Z1 = torch.matmul(torch.transpose(W1, 0, 1), X) + b1
A1 = torch.relu(Z1)
Z2 = torch.matmul(torch.transpose(W2, 0, 1), A1) + b2
output = torch.sigmoid(Z2)

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