NEURAL NETWORK 2

– Matee Vadrukchid –

Part 2: Multi Layer Neural Network (Backpropagation)

Part 2: Multi Layer Neural Network (Backpropagation)

$$\text{Output} = (x_1 \text{ AND } x_2) \text{ OR } (x_3 \text{ AND } x_4).$$

We'll focus on the **example**:

$$x_1 = 1, \quad x_2 = 0, \quad x_3 = 1, \quad x_4 = 1$$

$$y = (1 \wedge 0) \vee (1 \wedge 1) = 0 \vee 1 = 1.$$

Part 2: Multi Layer Neural Network (Backpropagation)

1. Network Structure

We have:

1. **Input layer**: 4 inputs (x_1, x_2, x_3, x_4).
2. **Hidden layer**: 2 neurons (call them h_1, h_2).
3. **Output layer**: 1 neuron (call it o).

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

We'll use the **sigmoid** activation function $\sigma(z) = \frac{1}{1+e^{-z}}$ in the hidden and output neurons.

Part 2: Multi Layer Neural Network (Backpropagation)

2. Initial Weights and Biases

Hidden neuron h_1:

- $w_{1,1}$ from x_1
- $w_{2,1}$ from x_2
- $w_{3,1}$ from x_3
- $w_{4,1}$ from x_4
- b_1 bias

Hidden neuron h_2:

- $w_{1,2}$ from x_1
- $w_{2,2}$ from x_2
- $w_{3,2}$ from x_3
- $w_{4,2}$ from x_4
- b_2 bias

Output neuron o:

- $w_{h1,o}$ from h_1
- $w_{h2,o}$ from h_2
- b_o bias

```
x1 \
x2  \
x3   >---> [ h1 ] ----------------\
x4 /                               \
                                  [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ----------------/
x4 /
```

```
h1:
   w_{1,1} = 0.10
   w_{2,1} = 0.20
   w_{3,1} = 0.30
   w_{4,1} = 0.40
   b_1     = 0.50

h2:
   w_{1,2} = 0.15
   w_{2,2} = 0.25
   w_{3,2} = 0.35
   w_{4,2} = 0.45
   b_2     = 0.55

Output neuron:
   w_{h1,o} = 0.60
   w_{h2,o} = 0.70
   b_o      = 0.80
```

Part 2: Multi Layer Neural Network (Backpropagation)

3. Forward Pass

Given our single training example:

$$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1, \quad \text{and target } y = 1.$$

3.1. Hidden Layer Computations

Neuron h_1

1. **Weighted sum** (z_1):

$$z_1 = w_{1,1}\, x_1 + w_{2,1}\, x_2 + w_{3,1}\, x_3 + w_{4,1}\, x_4 + b_1.$$

   Plugging in numbers:

$$z_1 = (0.10 \times 1) + (0.20 \times 0) + (0.30 \times 1) + (0.40 \times 1) + 0.50 = 0.10 + 0 + 0.30 + 0.40 + 0.50 = 1.30.$$

2. **Apply sigmoid** to get h_1:

$$h_1 = \sigma(z_1) = \frac{1}{1 + e^{-1.30}} \approx 0.7858 \text{ (approx)}.$$

Part 2: Multi Layer Neural Network (Backpropagation)

3. Forward Pass

Given our single training example:

$$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1, \quad \text{and target } y = 1.$$

Neuron h_2

1. **Weighted sum** (z_2):

   $$z_2 = (0.15 \times 1) + (0.25 \times 0) + (0.35 \times 1) + (0.45 \times 1) + 0.55 = 0.15 + 0 + 0.35 + 0.45 + 0.55 = 1.50.$$

2. **Apply sigmoid** to get h_2:

   $$h_2 = \sigma(z_2) = \frac{1}{1 + e^{-1.50}} \approx 0.8176.$$

Part 2: Multi Layer Neural Network (Backpropagation)

3. Forward Pass

Given our single training example:

$$x_1 = 1,\ x_2 = 0,\ x_3 = 1,\ x_4 = 1, \quad \text{and target } y = 1.$$

3.2. Output Neuron

Now the inputs are h_1 and h_2.

1. **Weighted sum** (z_o):

$$z_o = (h_1 \times w_{h1,o}) + (h_2 \times w_{h2,o}) + b_o.$$

   Numerically:

$$z_o = (0.7858 \times 0.60) + (0.8176 \times 0.70) + 0.80 = 0.4715 + 0.5723 + 0.80 = 1.8438 \text{ (approx)}.$$

2. **Apply sigmoid** to get final output $\hat{y}$:

$$\hat{y} = \sigma(z_o) = \frac{1}{1 + e^{-1.8438}} \approx 0.8634.$$

So the network's **prediction** is $\hat{y} \approx 0.8634$. The target is 1.

4. Compute Error

We'll use **Mean Squared Error** (for a single sample, it's just $\frac{1}{2}(\hat{y} - y)^2$):

$$\text{Error} = \frac{1}{2}(\hat{y} - y)^2 = \frac{1}{2}(0.8634 - 1)^2 = \frac{1}{2}(-0.1366)^2 = \frac{1}{2}(0.01866) \approx 0.00933.$$

We want this error to **go down** by adjusting weights.

Part 2: Multi Layer Neural Network (Backpropagation)

5. One Round of Backpropagation

We'll do these steps:

1. **Calculate derivatives** for the output layer weights & bias.
2. **Calculate derivatives** for the hidden layer weights & biases.
3. **Update** each weight/bias with the chosen learning rate η.

Let's pick a **learning rate** $\eta = 0.1$ (just as an example).

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                 [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

5.1. Output Layer Updates

We need the partial derivatives of the error w.r.t. each of $w_{h1,o}, w_{h2,o}, b_o$.

Part 2: Multi Layer Neural Network (Backpropagation)

5.1.1. Derivative for $w_{h1,o}$

Using the chain rule:

$$\frac{\partial \text{Error}}{\partial w_{h1,o}} = \frac{\partial \text{Error}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_o} \times \frac{\partial z_o}{\partial w_{h1,o}}.$$

1. $\frac{\partial \text{Error}}{\partial \hat{y}} = (\hat{y} - y) = 0.8634 - 1 = -0.1366.$
2. $\frac{\partial \hat{y}}{\partial z_o} = \hat{y}(1 - \hat{y}) \approx 0.8634 \times (1 - 0.8634) = 0.8634 \times 0.1366 \approx 0.1180.$
3. $\frac{\partial z_o}{\partial w_{h1,o}} = h_1 \approx 0.7858.$

Putting them together:

$$\frac{\partial \text{Error}}{\partial w_{h1,o}} = (-0.1366) \times (0.1180) \times (0.7858) \approx -0.0126 \quad \text{(negative value)}.$$

- Negative means we should **increase** $w_{h1,o}$ (since the derivative is negative, subtracting a negative leads to an increase).

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

Update rule (gradient descent):

$$w_{h1,o} \leftarrow w_{h1,o} - \eta \left(\frac{\partial \text{Error}}{\partial w_{h1,o}} \right) = 0.60 - 0.1 \times (-0.0126) = 0.60 + 0.00126 = 0.60126 \text{ (approx)}.$$

Part 2: Multi Layer Neural Network (Backpropagation)

5.1.2. Derivative for $w_{h2,o}$

Similarly:

$$\frac{\partial z_o}{\partial w_{h2,o}} = h_2 \approx 0.8176.$$

$$\frac{\partial \text{Error}}{\partial w_{h2,o}} = (-0.1366) \times (0.1180) \times (0.8176) \approx -0.0131.$$

Update:

$$w_{h2,o} \leftarrow 0.70 - 0.1 \times (-0.0131) = 0.70 + 0.00131 = 0.70131.$$

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                   [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

Part 2: Multi Layer Neural Network (Backpropagation)

5.1.3. Derivative for b_o

$\frac{\partial z_o}{\partial b_o} = 1$, so:

$$\frac{\partial \text{Error}}{\partial b_o} = (-0.1366) \times 0.1180 \times 1 \approx -0.0161.$$

Update:

$$b_o \leftarrow 0.80 - 0.1 \times (-0.0161) = 0.80 + 0.00161 = 0.80161.$$

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                   [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

Part 2: Multi Layer Neural Network (Backpropagation)

5.2. Hidden Layer Updates

We do the same chain rule for each weight going into h_1 and h_2. For instance, for h_1, we want $\frac{\partial \text{Error}}{\partial w_{1,1}}$, $\frac{\partial \text{Error}}{\partial w_{2,1}}$, etc.

5.2.1. Example: $\frac{\partial \text{Error}}{\partial w_{1,1}}$

We'll outline the chain rule:

$$\frac{\partial \text{Error}}{\partial w_{1,1}} = \frac{\partial \text{Error}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_o} \times \frac{\partial z_o}{\partial h_1} \times \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_{1,1}}.$$

1. $\frac{\partial \text{Error}}{\partial \hat{y}} = -0.1366.$
2. $\frac{\partial \hat{y}}{\partial z_o} = 0.1180.$
3. $\frac{\partial z_o}{\partial h_1} = w_{h1,o}^{(\text{old})}$. Typically we use the **old** value before this update step, which is 0.60.
4. $\frac{\partial h_1}{\partial z_1} = h_1(1 - h_1) \approx 0.7858 \times 0.2142 \approx 0.1684.$
   (Because $\sigma'(z) = \sigma(z) \times [1 - \sigma(z)]$.)
5. $\frac{\partial z_1}{\partial w_{1,1}} = x_1 = 1.$

```
x1 \
x2  \
x3   >---> [ h1 ] ---------------\
x4 /                              \
                                 [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ---------------/
x4 /
```

Multiply them all:

$$(-0.1366) \times (0.1180) \times (0.60) \times (0.1684) \times (1) \approx -0.00163 \text{ (approx)}.$$

Update rule ($\eta = 0.1$):

$$w_{1,1} \leftarrow 0.10 - 0.1 \times (-0.00163) = 0.10 + 0.000163 = 0.100163.$$

Part 2: Multi Layer Neural Network (Backpropagation)

5.2. Hidden Layer Updates

We do the same chain rule for each weight going into h_1 and h_2. For instance, for h_1, we want $\frac{\partial \text{Error}}{\partial w_{1,1}}$, $\frac{\partial \text{Error}}{\partial w_{2,1}}$, etc.

5.2.2. Other Weights into h_1

- $w_{2,1}$: Same chain rule, except $\frac{\partial z_1}{\partial w_{2,1}} = x_2 = 0$.
  - If $x_2 = 0$, then effectively $\frac{\partial \text{Error}}{\partial w_{2,1}} = 0$.
  - So $w_{2,1}$ will **not** change in this iteration!
- $w_{3,1}$: Same chain rule, but $\frac{\partial z_1}{\partial w_{3,1}} = x_3 = 1$. We'd get a similar numeric result as $w_{1,1}$.
- $w_{4,1}$: Similarly, $\frac{\partial z_1}{\partial w_{4,1}} = x_4 = 1$.
- b_1: $\frac{\partial z_1}{\partial b_1} = 1$. We multiply by the rest of the chain rule factors but not by any x_i.

```
x1 \
x2  \
x3   >---> [ h1 ] ----------------\
x4 /                               \
                                  [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ----------------/
x4 /
```

Part 2: Multi Layer Neural Network (Backpropagation)

5.2. Hidden Layer Updates

We do the same chain rule for each weight going into h_1 and h_2. For instance, for h_1, we want $\frac{\partial \text{Error}}{\partial w_{1,1}}$, $\frac{\partial \text{Error}}{\partial w_{2,1}}$, etc.

```
x1 \
x2  \
x3   >---> [ h1 ] ----------------\
x4 /                               \
                                 [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ----------------/
x4 /
```

5.2.3. Weights into h_2

Similarly for h_2, we have $w_{1,2}, w_{2,2}, w_{3,2}, w_{4,2}, b_2$. The chain rule is:

$$\frac{\partial \text{Error}}{\partial w_{1,2}} = \frac{\partial \text{Error}}{\partial \hat{y}} \times \frac{\partial \hat{y}}{\partial z_o} \times \frac{\partial z_o}{\partial h_2} \times \frac{\partial h_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_{1,2}}.$$

- $\frac{\partial z_o}{\partial h_2} = w_{h2,o}^{(\text{old})} = 0.70.$
- $\frac{\partial h_2}{\partial z_2} = h_2(1 - h_2) \approx 0.8176 \times 0.1824 \approx 0.1492.$
- $\frac{\partial z_2}{\partial w_{1,2}} = x_1 = 1.$

Then do the numeric multiplication and update. The same pattern for $w_{2,2}, w_{3,2}, w_{4,2}, b_2$.

Part 2: Multi Layer Neural Network (Backpropagation)

6. Conclusion After One Training Step

After performing these updates for **all** weights in the hidden and output layers, we get **slightly adjusted** weights. In summary:

- **Output layer**:
  - $w_{h1,o}$ goes from 0.60 to ≈ 0.60126.
  - $w_{h2,o}$ goes from 0.70 to ≈ 0.70131.
  - b_o goes from 0.80 to ≈ 0.80161.
- **Hidden layer** (example partial):
  - $w_{1,1}$ from 0.10 to ≈ 0.100163.
  - $w_{2,1}$ unchanged (because $x_2 = 0$).
  - $w_{3,1}, w_{4,1}, b_1$ also get small adjustments (not shown explicitly).
  - $w_{1,2}, w_{2,2}, w_{3,2}, w_{4,2}, b_2$ also get their own small adjustments.

```
x1 \
x2  \
x3   >---> [ h1 ] ----------------\
x4 /                               \
                                   [  o  ]  --> y_hat
x1 \
x2  \
x3   >---> [ h2 ] ----------------/
x4 /
```

If we do **many** passes (epochs) over different training examples (covering all possible 4-bit inputs) with the correct targets $(x_1 \wedge x_2) \vee (x_3 \wedge x_4)$, eventually the network's outputs can learn to approximate the **logical function**.

Part 2: Multi Layer Neural Network (Backpropagation)

Coding with Error

Using the updated error function $\mathrm{Error} = (\hat{y} - y)^2$: