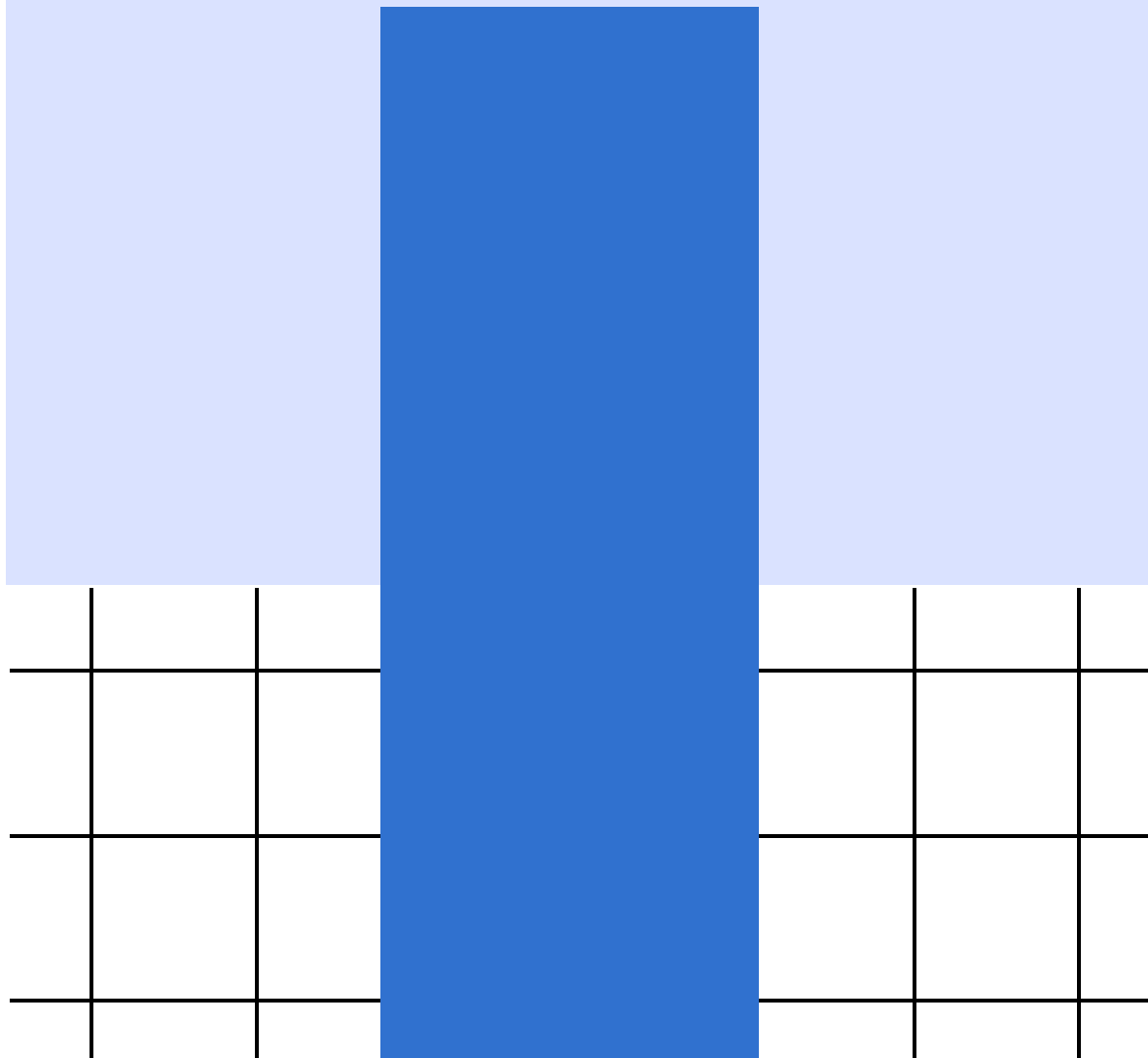
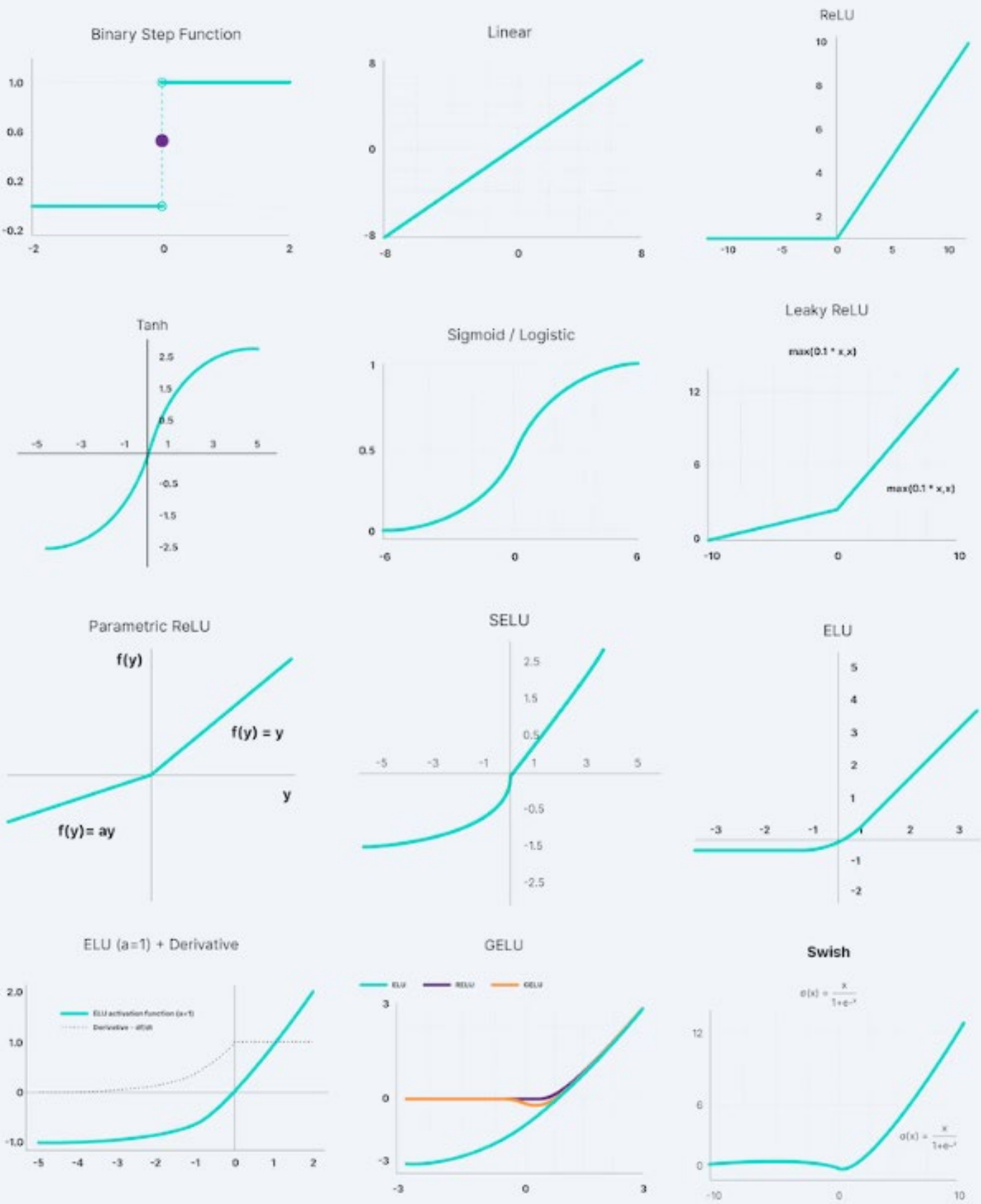


NEURAL NETWORK 1

– Matee Vadrukchid –



Neural Network Activation Functions



Function	Type	Forward	Backward
Linear	Many-Many	$\bar{z}_{i+1} = W\bar{z}_i$	$\bar{g}_i = W^T\bar{g}_{i+1}$
Sigmoid	One-One	$\bar{z}_{i+1} = \text{sigmoid}(\bar{z}_i)$	$\bar{g}_i = \bar{g}_{i+1} \odot \bar{z}_{i+1} \odot (\bar{1} - \bar{z}_{i+1})$
Tanh	One-One	$\bar{z}_{i+1} = \tanh(\bar{z}_i)$	$\bar{g}_i = \bar{g}_{i+1} \odot (\bar{1} - \bar{z}_{i+1} \odot \bar{z}_{i+1})$
ReLU	One-One	$\bar{z}_{i+1} = \bar{z}_i \odot I(\bar{z}_i > 0)$	$\bar{g}_i = \bar{g}_{i+1} \odot I(\bar{z}_i > 0)$
Hard Tanh	One-One	Set to ± 1 ($\notin [-1, +1]$) Copy ($\in [-1, +1]$)	Set to 0 ($\notin [-1, +1]$) Copy ($\in [-1, +1]$)
Max	Many-One	Maximum of inputs	Set to 0 (non-maximal inputs) Copy (maximal input)
Arbitrary function $f_k(\cdot)$	Anything	$\bar{z}_{i+1}^{(k)} = f_k(\bar{z}_i)$	$\bar{g}_i = J^T\bar{g}_{i+1}$ J is Jacobian (Equation 2.21)



Part 1: The Delta Rule (Single Neuron)

Part 1: The Delta Rule (Single Neuron)

Setup:

- Suppose we have a single neuron with inputs $\mathbf{x} = (x_1, x_2, \dots, x_n)$ and corresponding weights $\mathbf{w} = (w_1, w_2, \dots, w_n)$.
- The neuron's output (before activation) is:

$$z = \sum_{i=1}^n w_i x_i$$

- After applying an activation function $\phi(z)$ (for instance, a sigmoid), the final output is:

$$y = \phi(z)$$

Part 1: The Delta Rule (Single Neuron)

Error Function:

We define an error (or loss) function for a given training sample:

$$E = \frac{1}{2}(t - y)^2$$

where t is the target output and y is the neuron's actual output.

Goal:

We want to adjust the weights to minimize the error E .

Part 1: The Delta Rule (Single Neuron)

Deriving the Delta Rule:

To find how weights should change, we compute the gradient of E with respect to each weight w_i :

1. Compute the derivative of the error with respect to the output:

$$\frac{\partial E}{\partial y} = (y - t)$$

2. Compute the derivative of the output with respect to the net input z :

$$\frac{\partial y}{\partial z} = \phi'(z)$$

3. Compute the derivative of the net input with respect to the weight w_i :

$$\frac{\partial z}{\partial w_i} = x_i$$

Part 1: The Delta Rule (Single Neuron)

Chain these together:

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w_i} = (y - t)\phi'(z)x_i$$

Update Rule: We move the weights in the direction opposite to the gradient to reduce the error:

$$w_i^{new} = w_i^{old} - \eta \frac{\partial E}{\partial w_i} = w_i^{old} - \eta(y - t)\phi'(z)x_i$$

This is the **delta rule**. Often we define $\delta = (y - t)\phi'(z)$ and write:

$$w_i^{new} = w_i^{old} - \eta\delta x_i$$

Part 1: The Delta Rule (Single Neuron)

CODE

Scenario: We will train a single neuron to learn a simple mapping, say the logical function “AND”, using a differentiable activation (sigmoid). Although a single neuron cannot perfectly learn “AND” without a bias, let’s just assume we include a bias or choose a linearly separable case for demonstration (e.g., mapping a set of input values to a continuous target).

Note: Since we used a sigmoid activation and a linearly increasing target ($0.5 \cdot x$), the neuron’s output will be constrained between 0 and 1. So the model will learn a “squashed” approximation. To properly learn a scaled linear function, we might remove the sigmoid or use a linear activation for demonstration. But this code shows the delta rule in action.



Part 2: Backpropagation (Multi - Layer Network)



Part 2: Backpropagation (Multi -Layer Network)

General Steps:

- **Forward Pass:** For a two-layer network:

$$\text{Layer 1 (hidden): } z^{(1)} = W^{(1)}x + b^{(1)}, \quad h = \phi(z^{(1)})$$

$$\text{Layer 2 (output): } z^{(2)} = W^{(2)}h + b^{(2)}, \quad y = \phi(z^{(2)})$$

- **Compute Error:**

$$E = \frac{1}{2} \sum (t - y)^2$$

Part 2: Backpropagation (Multi -Layer Network)

- **Output Layer Delta:** For the output layer, similar to the single neuron:

$$\delta^{(2)} = (y - t)\phi'(z^{(2)})$$

- **Hidden Layer Delta:** The hidden layer delta is computed by propagating the output layer delta backwards:

$$\delta^{(1)} = (\delta^{(2)}W^{(2)T}) \odot \phi'(z^{(1)})$$

Part 2: Backpropagation (Multi -Layer Network)

- **Weight Updates:**

$$W^{(2)} = W^{(2)} - \eta \delta^{(2)} h^T$$

$$b^{(2)} = b^{(2)} - \eta \sum \delta^{(2)}$$

$$W^{(1)} = W^{(1)} - \eta \delta^{(1)} x^T$$

$$b^{(1)} = b^{(1)} - \eta \sum \delta^{(1)}$$

Part 2: Backpropagation (Multi -Layer Network)

- **Output Layer Delta:** For the output layer, similar to the single neuron:

$$\delta^{(2)} = (y - t)\phi'(z^{(2)})$$

Mathematical Derivation:

1. **Error function:**

Typically, we use:

$$E = \frac{1}{2}(t - y)^2$$

This function quantifies how "wrong" the output y is compared to t .

2. **Chain rule application:**

To find how to adjust weights, we need $\frac{\partial E}{\partial w}$. To get there, we start by looking at the intermediate terms:

$$\frac{\partial E}{\partial w} = \frac{\partial E}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Here:

- $\frac{\partial E}{\partial y} = (y - t)$ because if $E = \frac{1}{2}(t - y)^2$, then $\frac{\partial E}{\partial y} = (y - t)$.
- $\frac{\partial y}{\partial z} = \phi'(z)$, where $\phi(z)$ is the activation function. For a sigmoid, for instance, $\phi'(z) = y(1 - y)$.

Part 2: Backpropagation (Multi -Layer Network)

- **Hidden Layer Delta:** The hidden layer delta is computed by propagating the output layer delta backwards:

$$\delta^{(1)} = (\delta^{(2)} W^{(2)T}) \odot \phi'(z^{(1)})$$

Mathematical Reasoning (for a network with at least one hidden layer):

1. **Start from the output delta:**

We know the delta values at the output layer. The output error depends on the output layer's inputs, which in turn depend on the hidden layer's outputs. This means the hidden layer's contribution to the error must be inferred indirectly.

2. **Using the chain rule again:**

Consider a hidden neuron h_j in the hidden layer. Its activation h_j influences the output layer's input. Each output neuron o_k receives input from many hidden neurons:

$$z_k^{(2)} = \sum_j W_{jk}^{(2)} h_j + b_k^{(2)}$$

If we want $\frac{\partial E}{\partial h_j}$, we look at how changes in h_j affect the output error. Since the error at the output layer is known, we start from there:

$$\frac{\partial E}{\partial h_j} = \sum_k \frac{\partial E}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial h_j}$$

Part 2: Backpropagation (Multi -Layer Network)

- **Hidden Layer Delta:** The hidden layer delta is computed by propagating the output layer delta backwards:

$$\delta^{(1)} = (\delta^{(2)} W^{(2)T}) \odot \phi'(z^{(1)})$$

$$\frac{\partial E}{\partial h_j} = \sum_k \frac{\partial E}{\partial z_k^{(2)}} \cdot \frac{\partial z_k^{(2)}}{\partial h_j}$$

We already know $\frac{\partial E}{\partial z_k^{(2)}}$ is the delta at the output layer (for neuron k):

$$\delta_k^{(2)} = \frac{\partial E}{\partial z_k^{(2)}}$$

And $\frac{\partial z_k^{(2)}}{\partial h_j} = W_{jk}^{(2)}$, since $z_k^{(2)} = \sum_j W_{jk}^{(2)} h_j + b_k^{(2)}$.

Thus:

$$\frac{\partial E}{\partial h_j} = \sum_k \delta_k^{(2)} W_{jk}^{(2)}$$

Part 2: Backpropagation (Multi -Layer Network)

- **Hidden Layer Delta:** The hidden layer delta is computed by propagating the output layer delta backwards:

$$\delta^{(1)} = (\delta^{(2)} W^{(2)T}) \odot \phi'(z^{(1)})$$

3. From $\partial E / \partial h_j$ to $\delta_j^{(1)}$:

The hidden neuron's activation h_j is itself an output of its own activation function $\phi^{(1)}$:

$$h_j = \phi^{(1)}(z_j^{(1)})$$

We need the sensitivity with respect to $z_j^{(1)}$, not just h_j :

$$\frac{\partial E}{\partial z_j^{(1)}} = \frac{\partial E}{\partial h_j} \cdot \frac{\partial h_j}{\partial z_j^{(1)}} = \left(\sum_k \delta_k^{(2)} W_{jk}^{(2)} \right) \phi'(z_j^{(1)})$$

Part 2: Backpropagation (Multi -Layer Network)

- **Hidden Layer Delta:** The hidden layer delta is computed by propagating the output layer delta backwards:

$$\delta^{(1)} = (\delta^{(2)} W^{(2)T}) \odot \phi'(z^{(1)})$$

4. **Define the hidden delta:** We now have an expression for the hidden layer delta:

$$\delta_j^{(1)} = \left(\sum_k \delta_k^{(2)} W_{jk}^{(2)} \right) \cdot \phi'(z_j^{(1)})$$

This means the delta at a hidden neuron is the weighted sum of the deltas at the layer above (the output layer, in a 2-layer scenario), scaled by the derivative of the hidden neuron's activation function.

Network Structure

Architecture: A small neural network with:

- **Input layer:** 2 neurons (not counting bias)
- **Hidden layer:** 2 neurons
- **Output layer:** 1 neuron

Activation function: Let's use a sigmoid activation for the hidden and output neurons:

$$\phi(z) = \frac{1}{1 + e^{-z}}$$

We will also need its derivative:

$$\phi'(z) = \phi(z)(1 - \phi(z))$$

Given Values

Inputs: Suppose the input is $X = (x_1, x_2) = (0.5, 0.1)$.

Target: The desired output is $t = 0.8$.

Initial Weights:

- From Input to Hidden:

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix} = \begin{bmatrix} 0.10 & -0.20 \\ 0.40 & 0.10 \end{bmatrix}$$

Here $w_{11}^{(1)} = 0.10$ is the weight from input x_1 to hidden neuron 1, $w_{12}^{(1)} = -0.20$ is from x_1 to hidden neuron 2, etc.

- Hidden biases:

$$b^{(1)} = (b_1^{(1)}, b_2^{(1)}) = (0.0, 0.0)$$

For simplicity, let's assume no bias or zero bias in the hidden layer.

- From Hidden to Output:

$$W^{(2)} = \begin{bmatrix} w_{11}^{(2)} \\ w_{21}^{(2)} \end{bmatrix} = \begin{bmatrix} 0.30 \\ -0.10 \end{bmatrix}$$

- Output bias:

$$b^{(2)} = 0.0$$

Learning Rate: $\eta = 0.1$.

Forward Pass Computation

1. **Compute hidden layer inputs:** For hidden neuron 1:

$$z_1^{(1)} = w_{11}^{(1)}x_1 + w_{21}^{(1)}x_2 + b_1^{(1)} = (0.10)(0.5) + (0.40)(0.1) + 0.0 = 0.05 + 0.04 = 0.09$$

For hidden neuron 2:

$$z_2^{(1)} = w_{12}^{(1)}x_1 + w_{22}^{(1)}x_2 + b_2^{(1)} = (-0.20)(0.5) + (0.10)(0.1) + 0.0 = -0.10 + 0.01 = -0.09$$

2. **Hidden layer activations:** Apply the sigmoid:

$$h_1 = \phi(z_1^{(1)}) = \frac{1}{1 + e^{-0.09}} \approx 0.5225$$

(Here, $\phi(0.09)$ is roughly 0.5225 since $\phi(0) = 0.5$ and 0.09 is small.)

$$h_2 = \phi(z_2^{(1)}) = \frac{1}{1 + e^{0.09}} \approx 0.4775$$

(Since -0.09 is slightly negative, the sigmoid will be slightly less than 0.5.)

3. **Compute output layer input:**

$$z^{(2)} = w_{11}^{(2)} h_1 + w_{21}^{(2)} h_2 + b^{(2)} = (0.30)(0.5225) + (-0.10)(0.4775) + 0.0.$$

Calculate step-by-step:

$$(0.30)(0.5225) = 0.15675, \quad (-0.10)(0.4775) = -0.04775$$

$$z^{(2)} = 0.15675 - 0.04775 = 0.109$$

4. **Output activation:**

$$y = \phi(z^{(2)}) = \frac{1}{1 + e^{-0.109}} \approx 0.5272$$

So the network's output before training update is $y \approx 0.5272$.

Compute the Error

We use mean squared error (MSE) for this single sample:

$$E = \frac{1}{2}(t - y)^2 = \frac{1}{2}(0.8 - 0.5272)^2 = \frac{1}{2}(0.2728)^2 = \frac{1}{2}(0.0744) \approx 0.0372$$

Backward Pass (Compute Deltas)

1. **Output layer delta:** First, compute $\phi'(z^{(2)})$:

$$\phi'(z^{(2)}) = y(1 - y) = (0.5272)(1 - 0.5272) = 0.5272 \times 0.4728 \approx 0.2490$$

The output delta is:

$$\delta^{(2)} = (y - t)\phi'(z^{(2)}) = (0.5272 - 0.8)(0.2490) = (-0.2728)(0.2490) \approx -0.0679$$

2. **Hidden layer deltas:** We must propagate $\delta^{(2)}$ backward through the weights $W^{(2)}$.

For hidden neuron h_1 :

$$\frac{\partial E}{\partial z_1^{(1)}} = \delta_1^{(1)} = (\delta^{(2)} w_{11}^{(2)})\phi'(z_1^{(1)})$$

We know $\phi'(z_1^{(1)}) = h_1(1 - h_1) = 0.5225(1 - 0.5225) = 0.5225 \times 0.4775 \approx 0.2494$.

Thus:

$$\delta_1^{(1)} = (-0.0679)(0.30)(0.2494).$$

Calculate step by step:

$$(0.30)(0.2494) \approx 0.07482$$

$$\delta_1^{(1)} = (-0.0679)(0.07482) \approx -0.00507$$

For hidden neuron h_2 : Similarly:

$$\phi'(z_2^{(1)}) = h_2(1 - h_2) = 0.4775(1 - 0.4775) = 0.4775 \times 0.5225 \approx 0.2494 \text{ (coincidentally the same)}$$

$$\delta_2^{(1)} = (\delta^{(2)} w_{21}^{(2)}) \phi'(z_2^{(1)}) = (-0.0679)(-0.10)(0.2494).$$

$$(-0.0679)(-0.10) = 0.00679.$$

$$\delta_2^{(1)} = 0.00679 \times 0.2494 \approx 0.00169$$

Compute Weight Gradients

Output layer gradients:

- For $W_{11}^{(2)}$ (weight from h_1 to output):

$$\frac{\partial E}{\partial w_{11}^{(2)}} = \delta^{(2)} h_1 = (-0.0679)(0.5225) = -0.0355$$

- For $W_{21}^{(2)}$ (weight from h_2 to output):

$$\frac{\partial E}{\partial w_{21}^{(2)}} = \delta^{(2)} h_2 = (-0.0679)(0.4775) = -0.0324$$

- For output bias $b^{(2)}$:

$$\frac{\partial E}{\partial b^{(2)}} = \delta^{(2)} = -0.0679$$

Hidden layer gradients:

- For $w_{11}^{(1)}$ (weight from x_1 to h_1):

$$\frac{\partial E}{\partial w_{11}^{(1)}} = \delta_1^{(1)} x_1 = (-0.00507)(0.5) = -0.002535$$

- For $w_{21}^{(1)}$ (weight from x_2 to h_1):

$$\frac{\partial E}{\partial w_{21}^{(1)}} = \delta_1^{(1)} x_2 = (-0.00507)(0.1) = -0.000507$$

- For $w_{12}^{(1)}$ (weight from x_1 to h_2):

$$\frac{\partial E}{\partial w_{12}^{(1)}} = \delta_2^{(1)} x_1 = (0.00169)(0.5) = 0.000845$$

- For $w_{22}^{(1)}$ (weight from x_2 to h_2):

$$\frac{\partial E}{\partial w_{22}^{(1)}} = \delta_2^{(1)} x_2 = (0.00169)(0.1) = 0.000169$$

- Hidden biases are zero in this example, so no bias gradient there.

Weight Updates

Use $\eta = 0.1$. The update rule:

$$w \leftarrow w - \eta \frac{\partial E}{\partial w}, \quad b \leftarrow b - \eta \frac{\partial E}{\partial b}$$

- Update output layer weights:

$$w_{11}^{(2)} \leftarrow 0.30 - 0.1(-0.0355) = 0.30 + 0.00355 = 0.30355$$

$$w_{21}^{(2)} \leftarrow (-0.10) - 0.1(-0.0324) = -0.10 + 0.00324 = -0.09676$$

$$b^{(2)} \leftarrow 0.0 - 0.1(-0.0679) = 0.0 + 0.00679 = 0.00679$$

- Update hidden layer weights:

$$w_{11}^{(1)} \leftarrow 0.10 - 0.1(-0.002535) = 0.10 + 0.0002535 = 0.1002535$$

$$w_{21}^{(1)} \leftarrow 0.40 - 0.1(-0.000507) = 0.40 + 0.0000507 = 0.4000507$$

$$w_{12}^{(1)} \leftarrow (-0.20) - 0.1(0.000845) = -0.20 - 0.0000845 = -0.2000845$$

$$w_{22}^{(1)} \leftarrow 0.10 - 0.1(0.000169) = 0.10 - 0.0000169 = 0.0999831$$

No hidden layer bias updates since biases were zero and also their gradient would be:

$$\frac{\partial E}{\partial b_1^{(1)}} = \delta_1^{(1)} \approx -0.00507, \quad b_1^{(1)} \leftarrow 0.0 - 0.1(-0.00507) = 0.000507$$

$$\frac{\partial E}{\partial b_2^{(1)}} = \delta_2^{(1)} \approx 0.00169, \quad b_2^{(1)} \leftarrow 0.0 - 0.1(0.00169) = -0.000169$$

(If we consider them. Initially, we said no bias or zero bias. If we allow updating them, we now have new non-zero biases. If the original instruction was zero bias and not to update them, you can skip this step. Otherwise, we have them now.)

Part 2: Backpropagation (Multi -Layer Network)

CODE

Scenario:

Let's create a small MLP to learn the XOR function, which is a classic problem not solvable by a single-layer perceptron alone. The XOR truth table:

x1	x2	XOR
0	0	0
0	1	1
1	0	1
1	1	0

We will use a hidden layer with sigmoid activation and an output layer with sigmoid activation.