Neural Networks

By Trần Minh Dương - Learning Support

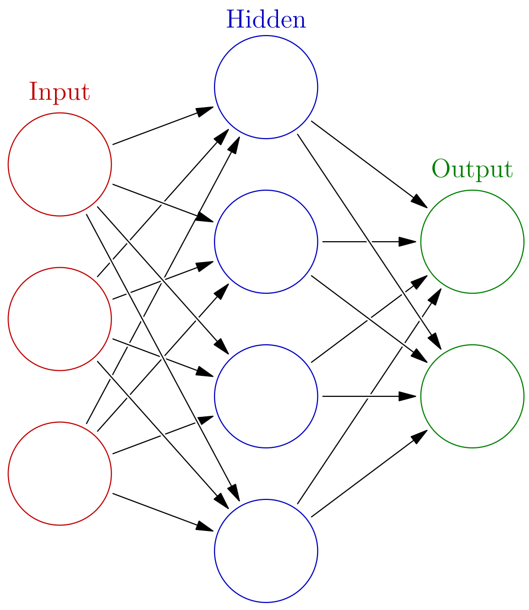
Overview

Neural networks are powerful models inspired by the human brain, capable of learning complex patterns from data. They consist of interconnected layers of neurons that process information and transform inputs into meaningful outputs. From image recognition to natural language processing, this document aims to uncover how neural networks are trained.

1. Architecture

A typical neural network is composed of the following:

- **Input Layer**: Receives raw data (features) and feeds it into the network.
- **Hidden Layers**: Performs computations with weights, biases, and activation functions to uncover patterns in the data.
- **Output Layer**: Produces the final result, such as a class label or prediction value.

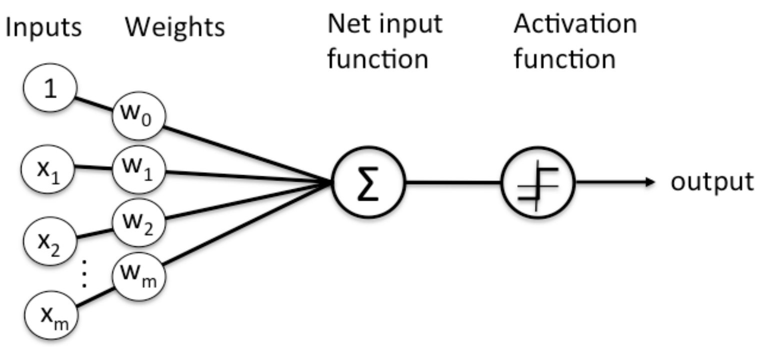


2. Learning Process

The network trains through these steps:

1. **Forward Pass**: Let data flow through the network, producing predictions.
2. **Loss Calculation**: A loss function measures prediction errors.
3. **Back Propagation**: Errors are propagated backward to adjust weights and biases.
4. **Optimization**: Algorithms like Gradient Descent minimize loss, improving predictions.

3. Forward Pass



To calculate the next node in a neural network, we do the following:

- Calculate the weighted sum of the previous nodes x_i:

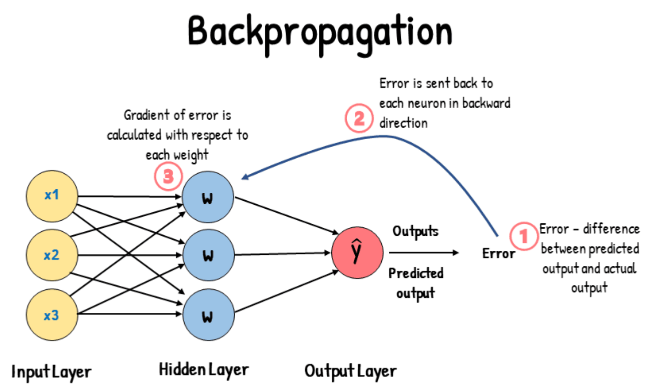
$$x_j = \sum_i x_i \cdot w_{ij} = x_1 \cdot w_{1j} + x_2 \cdot w_{2j} + \ldots$$

- Pass it through the activation function:

$$y_j = f(x_j)$$

To further understand, see exercise below

4.Backpropagation (Mad Difficult)


Backpropagation
Error is sent back to each neuron in backward direction
Gradient of error is calculated with respect to each weight
x1
x2
x3
w
w
w
ŷ
Outputs
Predicted output
Error
Error – difference between predicted output and actual output
Input Layer
Hidden Layer
Output Layer

Step 1. Error Calculation

The loss function measures the difference between the predicted output and the actual output:

$$\text{Loss} = \frac{1}{2} \cdot (y_{\text{output}} - y_{\text{actual}})^2$$

(Note that this loss function varies for different problems: A regression problem may use MSE while a classification one may use log-likelihood.)

Step 2. Error Signal for the Output Layer

The error signal at the output layer neuron is:

$$\delta_{\text{output}} = \frac{\partial \text{Loss}}{\partial x_{\text{output}}} = \frac{\partial \text{Loss}}{\partial y_{\text{output}}} \cdot \frac{\partial y_{\text{output}}}{\partial x_{\text{output}}} = (y_{\text{output}} - y_{\text{actual}}) \cdot F'(x_{\text{output}})$$

Step 3. Error Signal for Hidden Layer Neurons

The error signal for a hidden layer neuron j is backpropagated from the next layer:

$$\delta_j = \frac{\partial L}{\partial x_j} = \sum_{k \in \text{next layer}} \frac{\partial L}{\partial x_k} \cdot \frac{\partial x_k}{\partial y_j} \cdot \frac{\partial y_j}{\partial x_j} = \sum_{k \in \text{next layer}} \delta_k \cdot w_{jk} \cdot F'(x_j)$$

Where:

- δ_k is the error signal from the next layer.
- w_{jk} is the weight connecting neuron j to neuron k.
- $F'(x_j)$ is the derivative of the activation function of the hidden neuron.

Step 4. Weight Update Rule

The weights are updated to minimize the loss using the gradient descent formula:

$$w_{ij}^{\text{new}} = w_{ij}^{\text{old}} - \eta \cdot \delta_j \cdot y_i$$

Where:

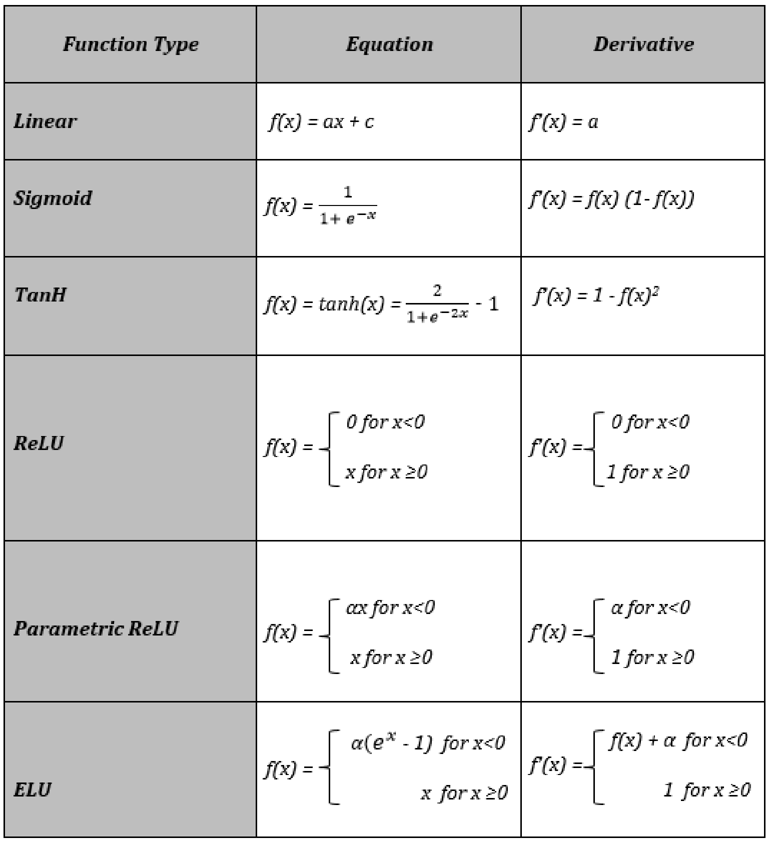
- η is the learning rate.
- δ_j is the error signal of neuron j.
- y_i is the output of the previous neuron (input to the current layer).

Step 5. Bias Update Rule (if applicable)

Biases are updated similarly to weights:

$$b_j^{\text{new}} = b_j^{\text{old}} - \eta \cdot \delta_j$$

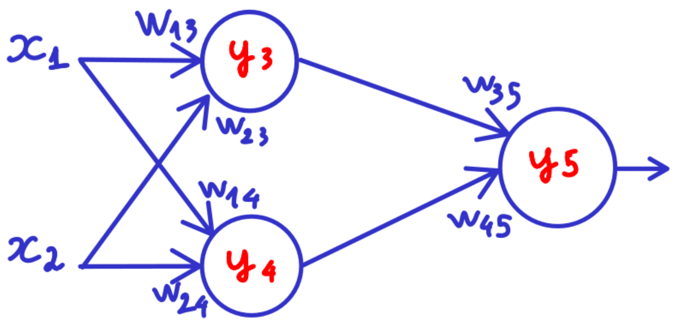
Activation Function Derivative



Function Type	**Equation**	**Derivative**
Linear	$f(x) = ax + c$	$f'(x) = a$
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)\,(1 - f(x))$
TanH	$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ReLU	$f(x) = \begin{cases} 0 \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
Parametric ReLU	$f(x) = \begin{cases} \alpha x \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$
ELU	$f(x) = \begin{cases} \alpha(e^{x} - 1) \text{ for } x<0 \\ x \text{ for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha \text{ for } x<0 \\ 1 \text{ for } x \geq 0 \end{cases}$

Exercise

Given the following neural network:



Weights:

$w_{13} = 2, w_{23} = -3$

$w_{14} = 1, w_{24} = 4$

$w_{35} = 2, w_{45} = -1$

The activation function is ReLU:

$$F(v) = \begin{cases} v & \text{if } v \geq 0 \\ 0 & \text{if } v < 0 \end{cases}$$

Derivative of ReLU

$$F'(v) = \begin{cases} 1 & \text{if } v > 0 \\ 0 & \text{if } v \leq 0 \end{cases}$$

Questions:

a) Predict the output with inputs: $x_1 = 1$ and $x_2 = -1$

b) Knowing that the actual output is y = 0, the loss function is given as:

$$\text{Loss} = \frac{1}{2} \cdot (y_{\text{output}} - y_{\text{actual}})^2$$

Using back-propagation, evaluate the weights after one iteration with $\eta = 0.1$.

Solution:

a) Predict the output with inputs: $x_1 = 1$ and $x_2 = -1$

Forward pass:

1. **At the node y_3:**

$$x_3 = x_1 \cdot w_{13} + x_2 \cdot w_{23} = 1 \cdot 2 + (-1) \cdot (-3) = 5$$

$$y_3 = F(x_3) = 5$$

2. **At the node y_4:**

$$x_4 = x_1 \cdot w_{14} + x_2 \cdot w_{24} = 1 \cdot 1 + (-1) \cdot 4 = -3$$

$$y_4 = F(x_4) = 0$$

3. **At the node y_5:**

$$x_5 = y_3 \cdot w_{35} + y_4 \cdot w_{45} = 5 \cdot 2 + 0 \cdot (-1) = 10$$

$$y_5 = F(x_5) = 10$$

Prediction:
The output of the neural network is:

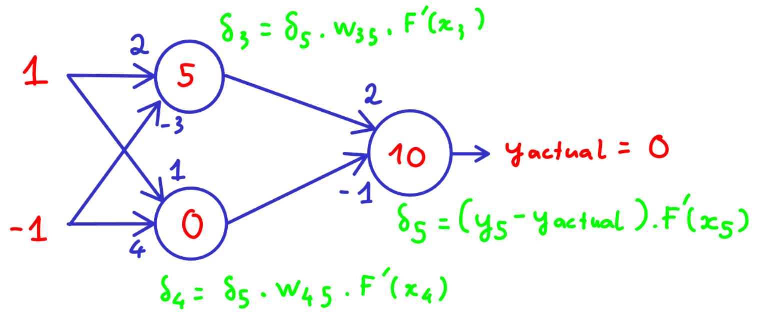
$$y_{\text{output}} = y_5 = 10$$

---

b) Back-propagation

$$F'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

$$\text{Update: } w_{ij}^{new} = w_{ij}^{old} - \eta \cdot \delta_j \cdot y_i$$



$\delta_3 = \delta_5 \cdot w_{35} \cdot F'(x_3)$
1
-1
2
-3
1
4
5
0
2
-1
10
$y_{actual} = 0$
$\delta_5 = (y_5 - y_{actual}) \cdot F'(x_5)$
$\delta_4 = \delta_5 \cdot w_{45} \cdot F'(x_4)$

Given:

- Actual output: $y = 0$
- Loss function:

$$\text{Loss} = \frac{1}{2} \cdot (y_{\text{output}} - y_{\text{actual}})^2$$

Error at the output node y_5:

$$\delta_5 = \frac{\partial \text{Loss}}{\partial x_5} = \frac{\partial \text{Loss}}{\partial y_5} \cdot \frac{\partial y_5}{\partial x_5} = (y_5 - y_{\text{actual}}) \cdot F'(x_5)$$

$$\delta_5 = (10 - 0) \cdot F'(10) = 10$$

Update weights w_{35} and w_{45}:

$$w_{35} = w_{35} - \eta \cdot \delta_5 \cdot y_3 = 2 - 0.1 \cdot 10 \cdot 5 = -3$$

$$w_{45} = w_{45} - \eta \cdot \delta_5 \cdot y_4 = -1 - 0.1 \cdot 10 \cdot 0 = -1$$

Error at the node y_3:

$$\delta_3 = \frac{\partial \text{Loss}}{\partial x_3} = \frac{\partial \text{Loss}}{\partial x_5} \cdot \frac{\partial x_5}{\partial y_3} \cdot \frac{\partial y_3}{\partial x_3} = \delta_5 \cdot w_{35} \cdot F'(x_3)$$

$$\delta_3 = 10 \cdot 2 \cdot F'(5) = 20$$

Update weights w_{13} and w_{23}:

$$w_{13} = w_{13} - \eta \cdot \delta_3 \cdot x_1 = 2 - 0.1 \cdot 20 \cdot 1 = 0$$

$$w_{23} = w_{23} - \eta \cdot \delta_3 \cdot x_2 = -3 - 0.1 \cdot 20 \cdot (-1) = -1$$

Error at the node y_4:

$$\delta_4 = \frac{\partial \text{Loss}}{\partial x_4} = \frac{\partial \text{Loss}}{\partial x_5} \cdot \frac{\partial x_5}{\partial y_4} \cdot \frac{\partial y_4}{\partial x_4} = \delta_5 \cdot w_{45} \cdot F'(x_4)$$

$$\delta_4 = 10 \cdot (-1) \cdot F'(-3) = 0$$

Update weights w_{14} and w_{24}:

$$w_{14} = w_{14} - \eta \cdot \delta_4 \cdot x_1 = 1 - 0.1 \cdot 0 \cdot 1 = 1$$

$$w_{24} = w_{24} - \eta \cdot \delta_4 \cdot x_2 = 4 - 0.1 \cdot 0 \cdot (-1) = 4$$

Final updated weights:

- $w_{13} = 0,\ w_{23} = -1$
- $w_{14} = 1,\ w_{24} = 4$
- $w_{35} = -3,\ w_{45} = -1$

In [1]:
```
import numpy as np

# Weights
weights = {
    "w13": 2, "w23": -3,
    "w14": 1, "w24": 4,
    "w35": 2, "w45": -1
}

# Activation function
def relu(x):
    return np.maximum(0, x)

def relu_derivative(x):
    return x>0

x1, x2 = 1, -1
y_actual = 0
eta = 0.1

# Forward pass
def forward_pass(weights, x1, x2):
    x3 = x1 * weights["w13"] + x2 * weights["w23"]
    y3 = relu(x3)

    x4 = x1 * weights["w14"] + x2 * weights["w24"]
    y4 = relu(x4)

    x5 = y3 * weights["w35"] + y4 * weights["w45"]
    y5 = relu(x5)

    return {"x3": x3, "y3": y3, "x4": x4, "y4": y4, "x5": x5, "y5": y5}
```

```python
def back_propagation(weights, forward_values, x1, x2, y_actual, eta):
    y5 = forward_values["y5"]
    x3, y3 = forward_values["x3"], forward_values["y3"]
    x4, y4 = forward_values["x4"], forward_values["y4"]
    x5 = forward_values["x5"]

    # Compute error signals
    delta5 = (y5 - y_actual) * relu_derivative(x5)
    delta3 = delta5 * weights["w35"] * relu_derivative(x3)
    delta4 = delta5 * weights["w45"] * relu_derivative(x4)

    # Update weights
    weights["w35"] -= eta * delta5 * y3
    weights["w45"] -= eta * delta5 * y4

    weights["w13"] -= eta * delta3 * x1
    weights["w23"] -= eta * delta3 * x2

    weights["w14"] -= eta * delta4 * x1
    weights["w24"] -= eta * delta4 * x2

    return weights

forward_values = forward_pass(weights, x1, x2)
print("Forward pass outputs:", forward_values)

updated_weights = back_propagation(weights, forward_values, x1, x2, y_actual, eta)
print("Updated weights:", updated_weights)
```

```
Forward pass outputs: {'x3': 5, 'y3': 5, 'x4': -3, 'y4': 0, 'x5': 10, 'y5': 10}
Updated weights: {'w13': 0.0, 'w23': -1.0, 'w14': 1.0, 'w24': 4.0, 'w35': -3.0, 'w45':
-1.0}
```

This document was created in Jupyter Notebook by Trần Minh Dương (tmd).

If you have any questions or notice any errors, feel free to reach out via Discord at @tmdhoctiengphap or @ICT-Supporters on the USTH Learning Support server.

Check out my GitHub repository for more projects: GalaxyAnnihilator/MachineLearning .