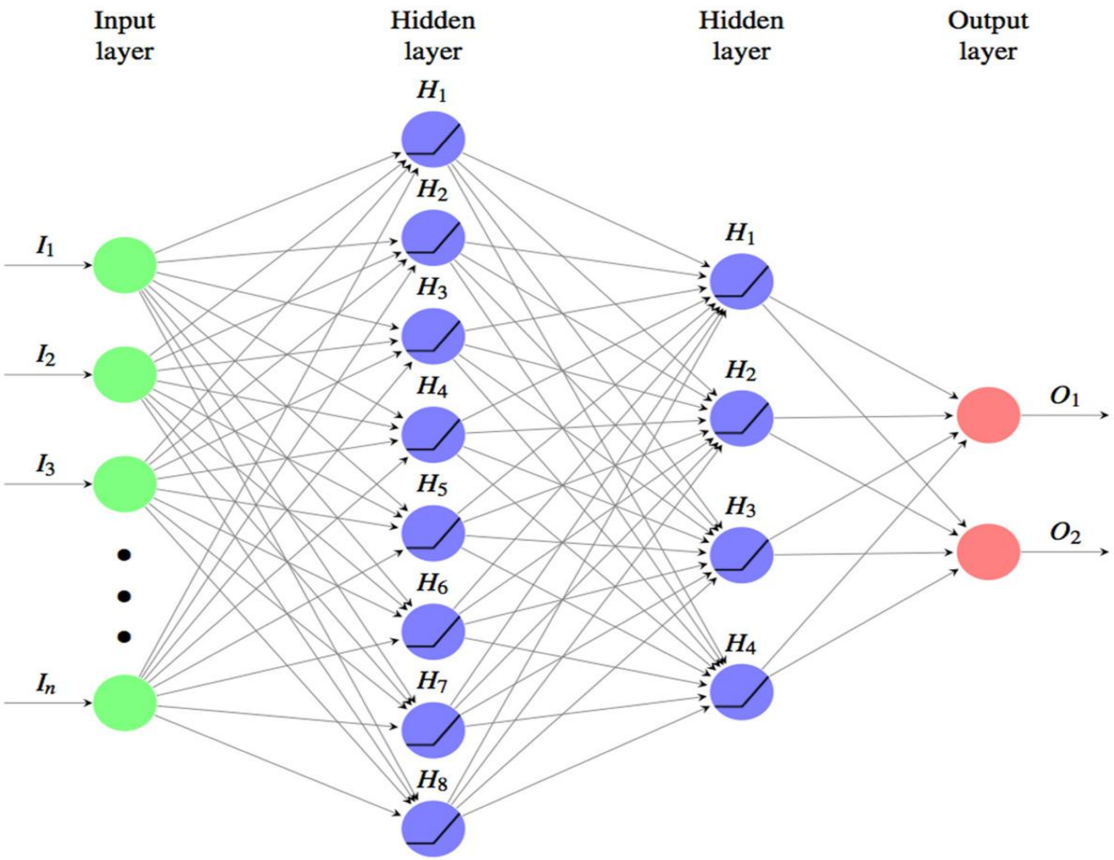
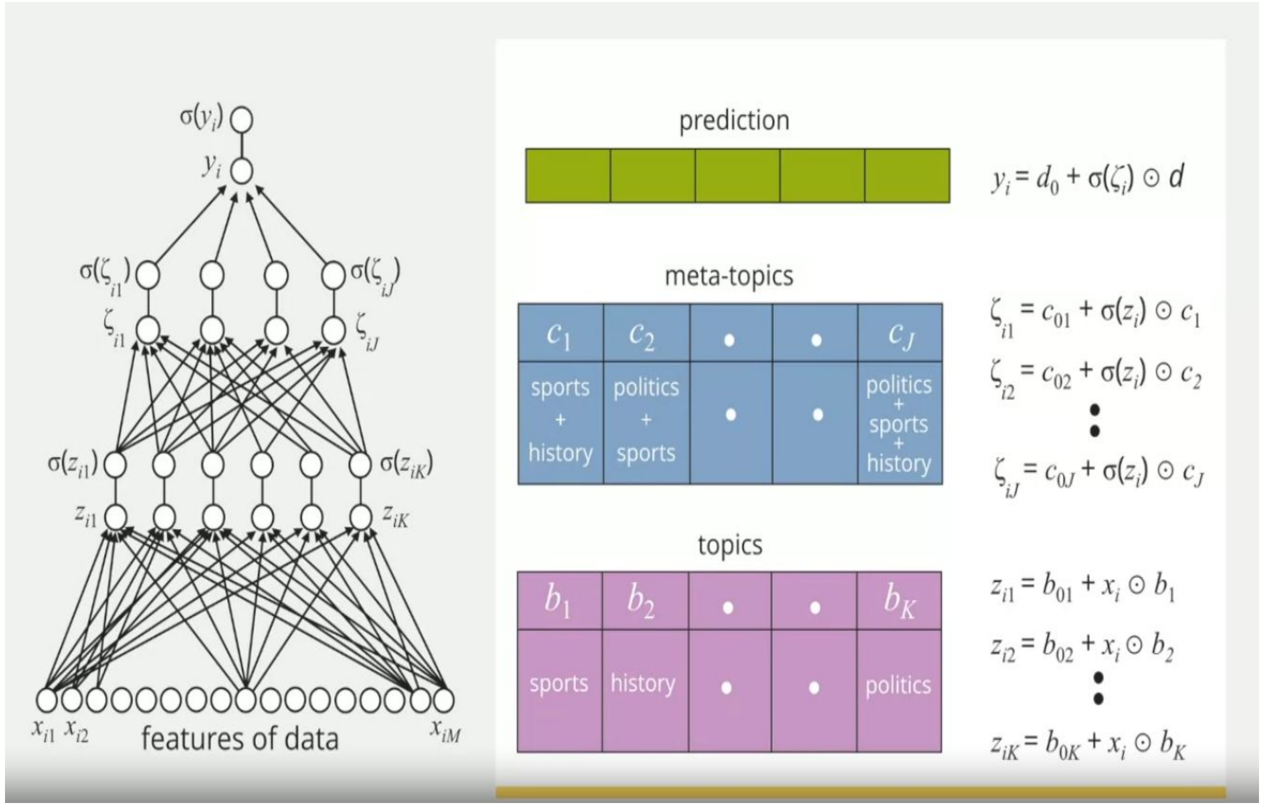
Types of Neural Networks

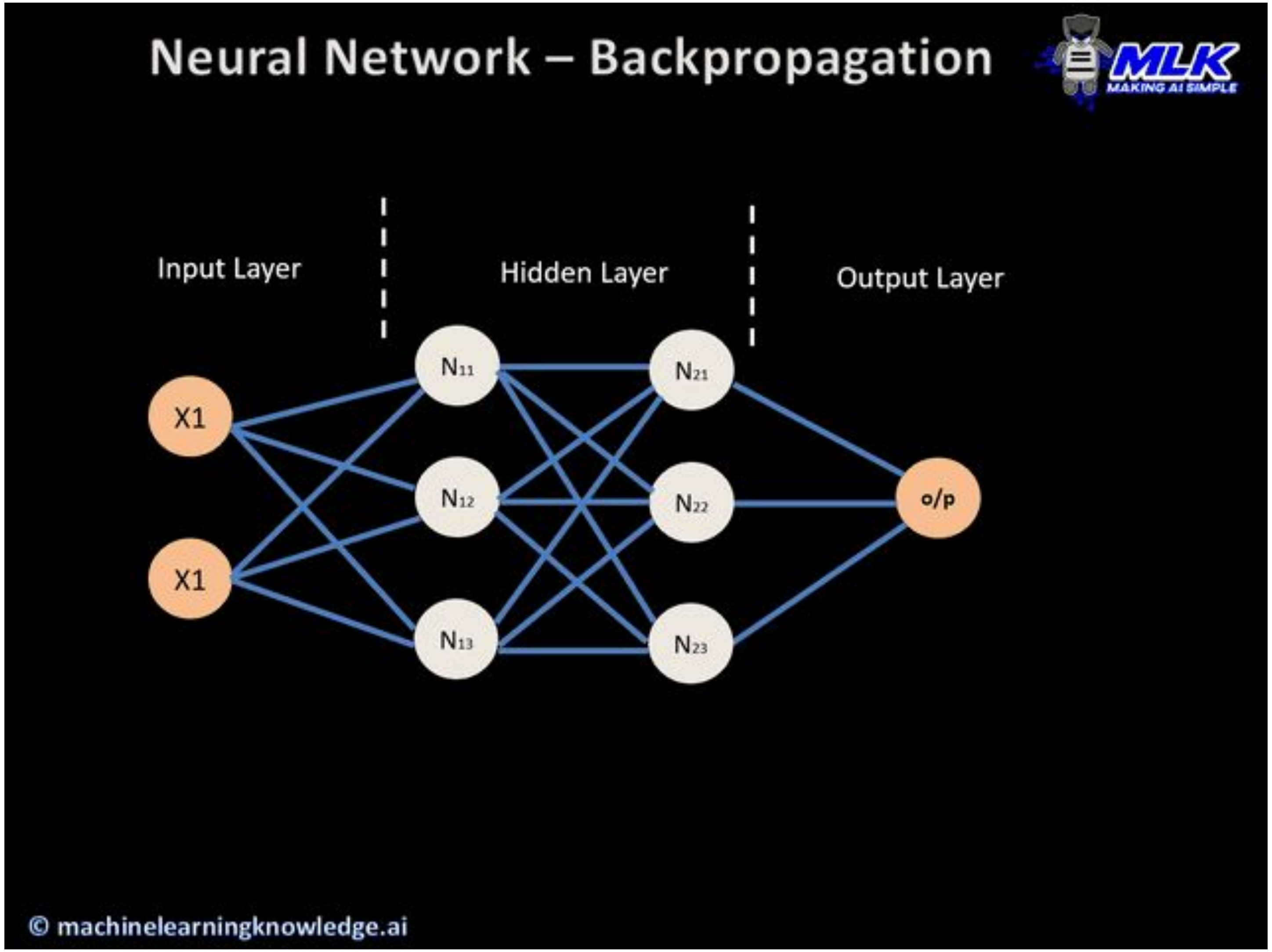
1. Feedforward Neural Networks (FNN)
2. Convolutional Neural Networks (CNN)
3. Recurrent Neural Networks (RNN)
4. Generative Adversarial Networks (GAN)
5. Autoencoders
6. Transformer Networks





Feedforward Neural Networks (FNN)

Basics of Neural Networks

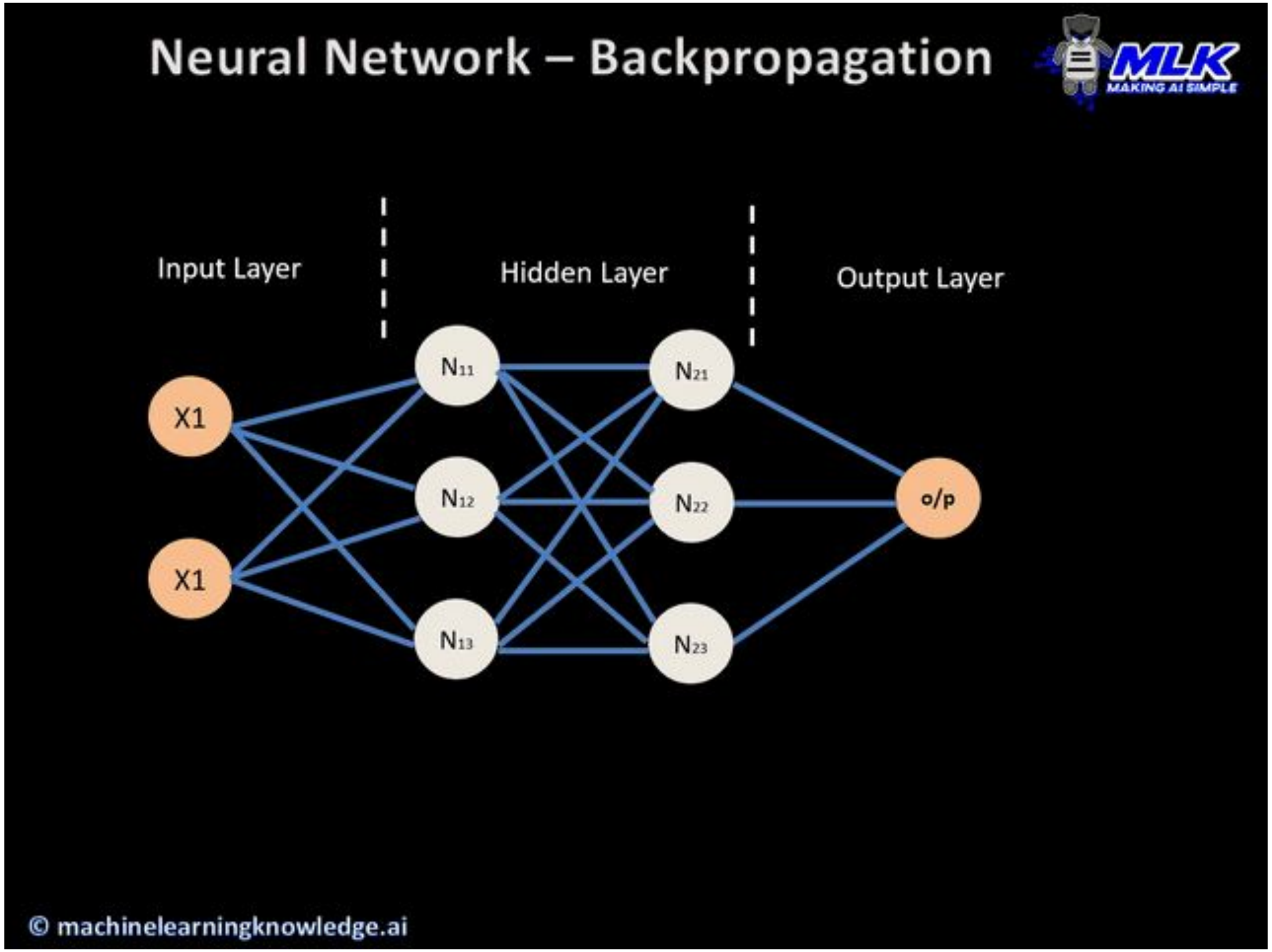


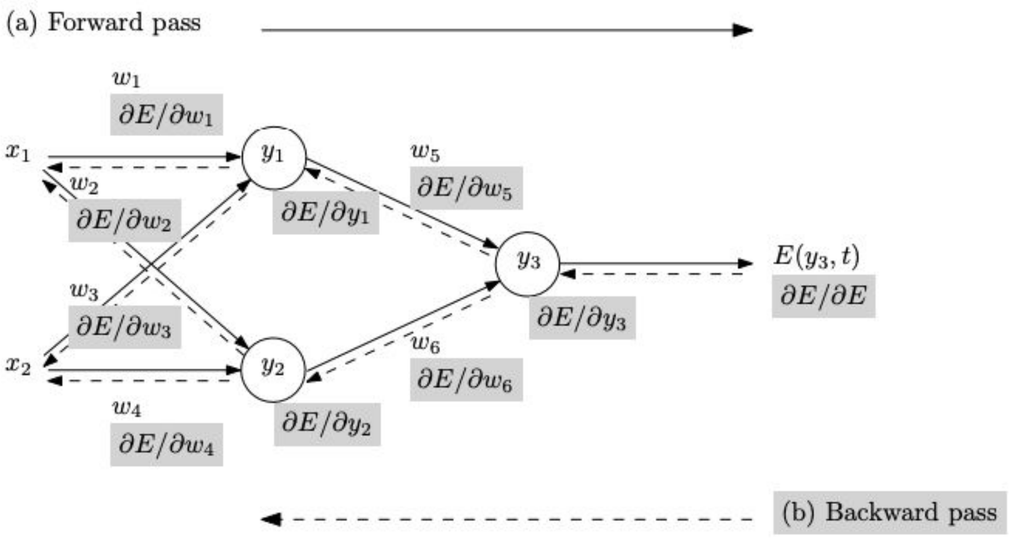
1. **Forward Pass:**
   - Compute the output of the network for a given input.
   - Calculate the cost (loss) using the cost function.

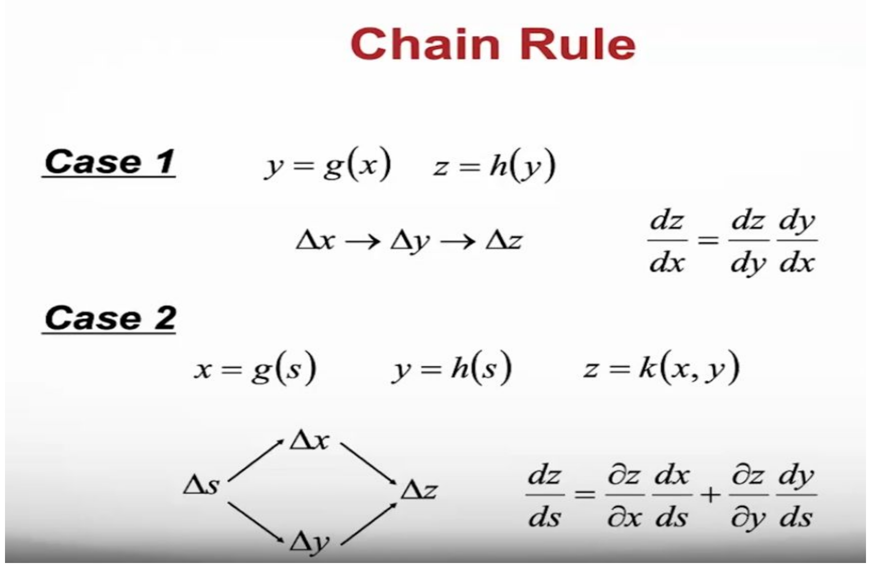
2. **Backward Pass:**
   - Compute the gradient of the loss function concerning the output of the network.
   - Propagate the gradients back through the network to compute the gradients concerning each weight and bias.

3. **Weight Update:**
   - Adjust the weights and biases using the gradient descents.

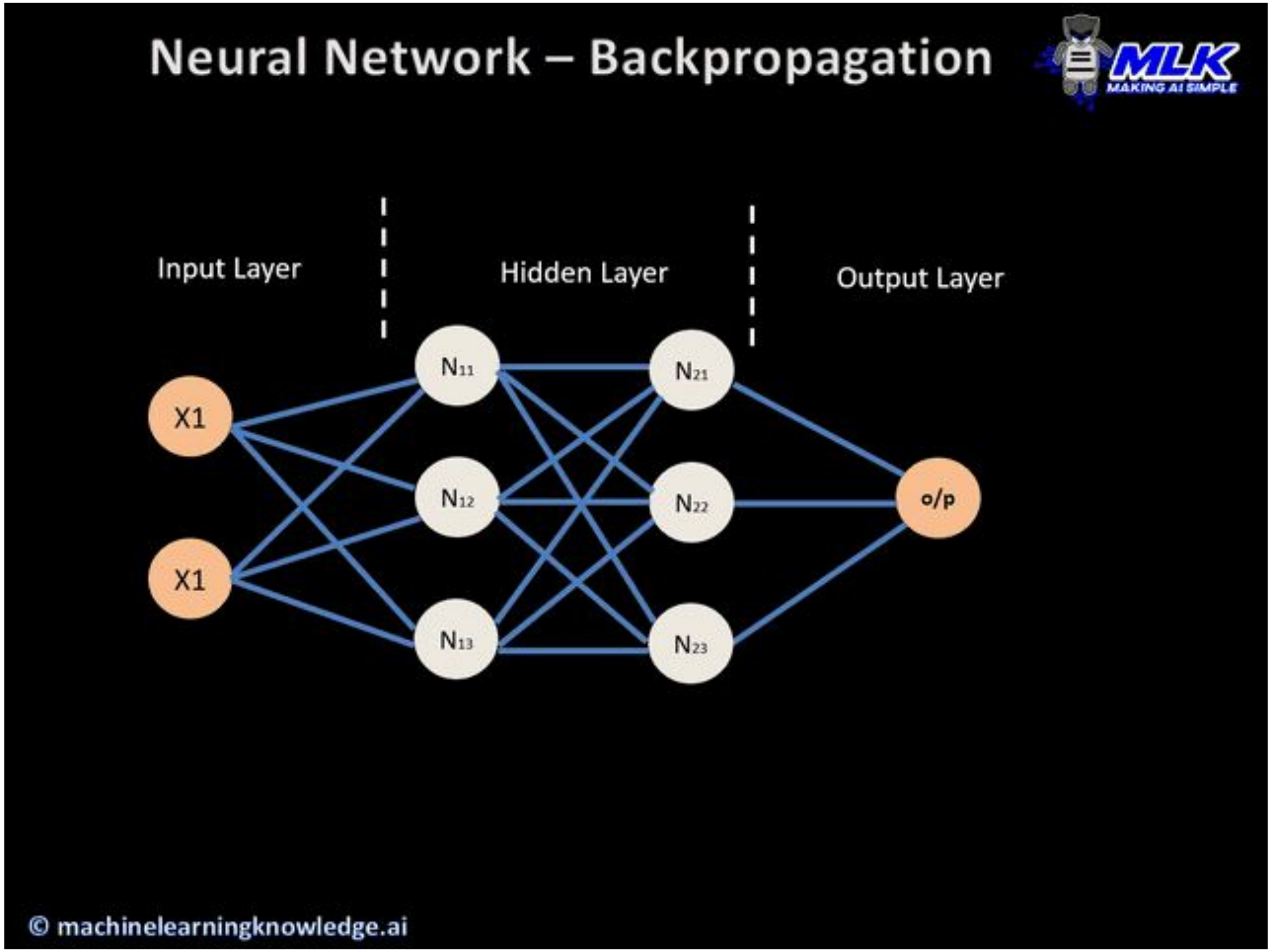
Basics of Neural Networks





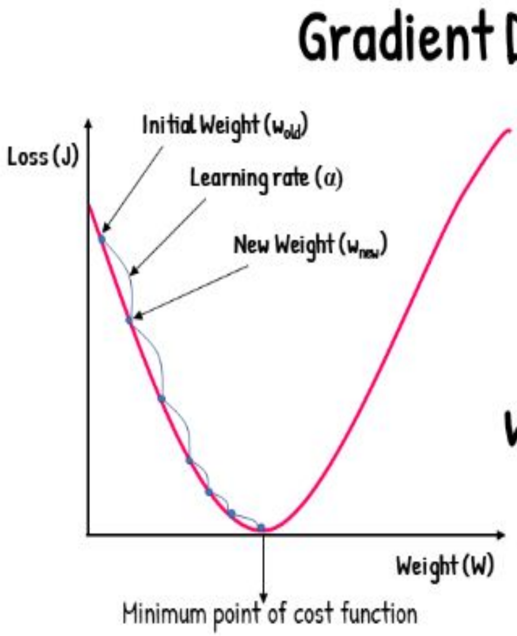


Basics of Neural Networks



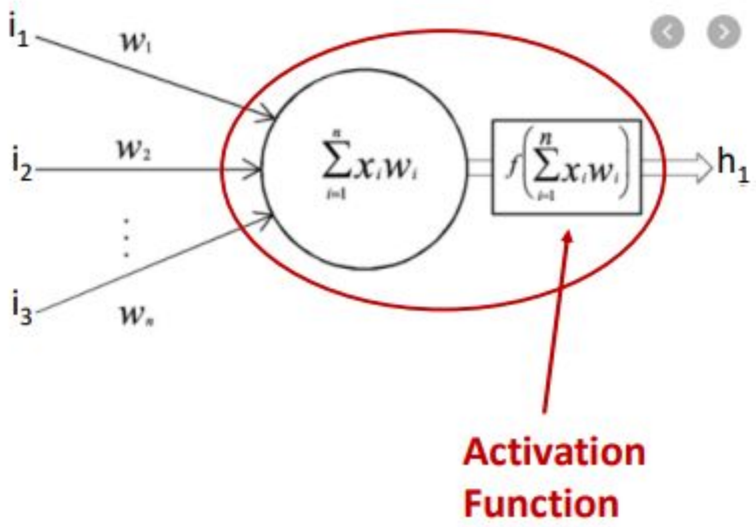
Weight Update:

- Adjust the weights and biases using the gradient descents.



$$W_{new} = W_{old} - \alpha \frac{\delta J}{\delta W}$$

Activation Functions



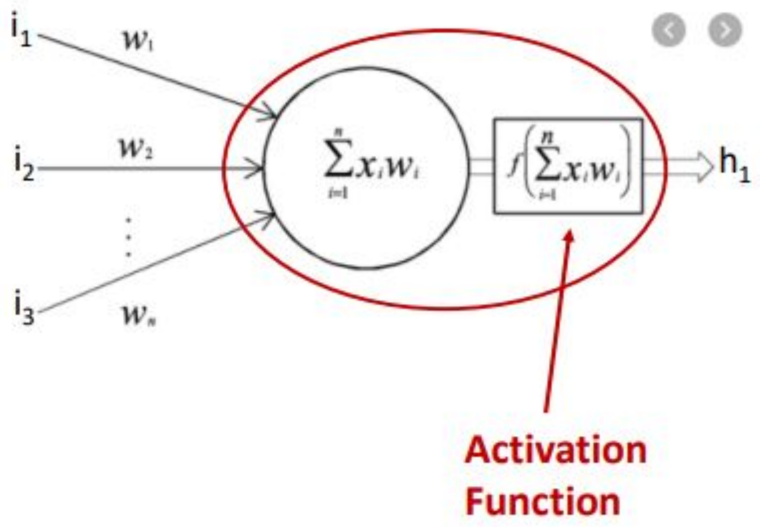
Activation Function is applied over the linear weighted summation of the incoming information to a node.

Convert linear input signals from perceptron to a linear/non-linear output signal.

It decides whether to activate a node or not.

Activation Functions

Activation functions must be monotonic, differentiable, and quickly converging.



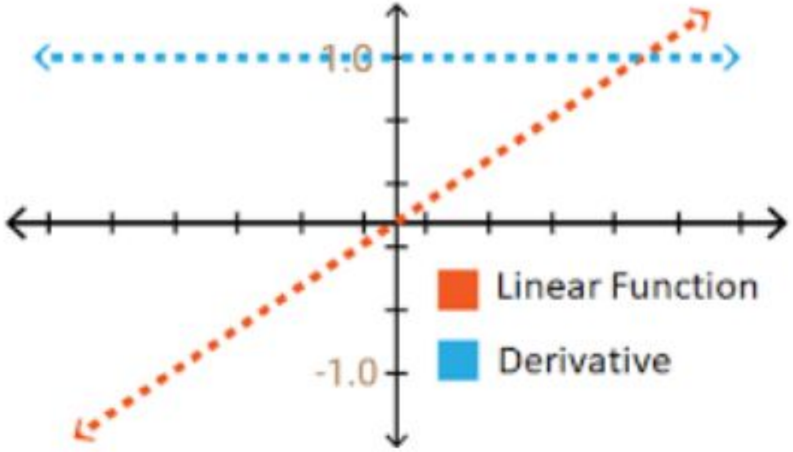
Types of Activation Functions:

- Linear
- Non-Linear

Linear

$$f(x) = ax + b$$

$$\frac{df(x)}{dx} = a$$



Observations:

- Constant gradient
- Gradient does not depend on the change in the input

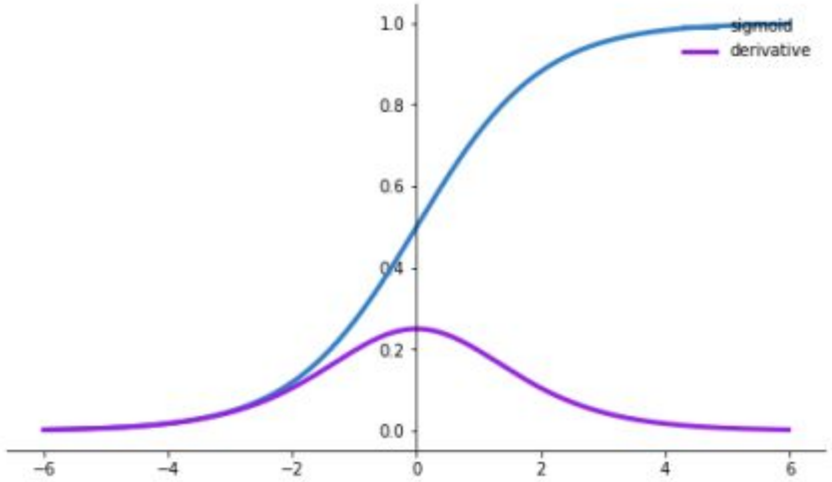
Non-Linear

- Sigmoid (Logistic)
- Hyperbolic Tangent (Tanh)
- Rectified Linear Unit (ReLU)
  - *Leaky Relu*
  - *Parametric Relu*
- Exponential Linear Unit (ELU)

Sigmoid Activation Functions (Logistics)

$$f(x) = \frac{1}{1 + e^{-x}}$$

$$\frac{df(x)}{dx} = f(x)(1 - f(x))$$



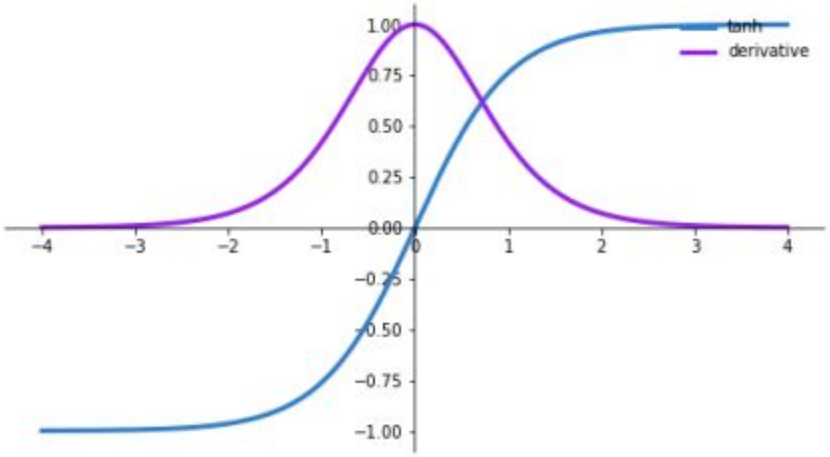
Observations:

- Output: 0 to 1
- Outputs are not zero-centered
- Can saturate and kill (vanish) gradients

Tanh Activation Function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

$$\frac{df(x)}{dx} = 1 - f(x)^2$$



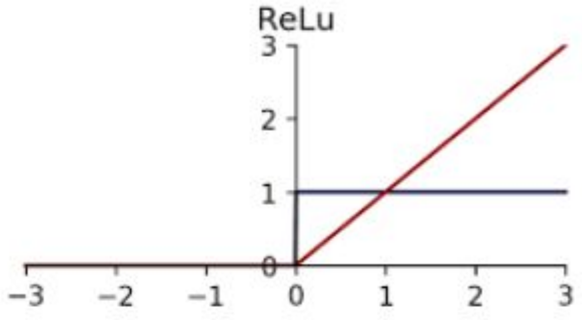
Observations:

- Output: -1 to +1
- Outputs are zero-centered
- Can Saturate and kill (vanish) gradients
- Gradient is more steeped than Sigmoid, resulting in faster convergence

Rectified Linear Unit(ReLU)

$$f(x) = \max(0, x)$$

$$\frac{df(x)}{dx} = 1$$



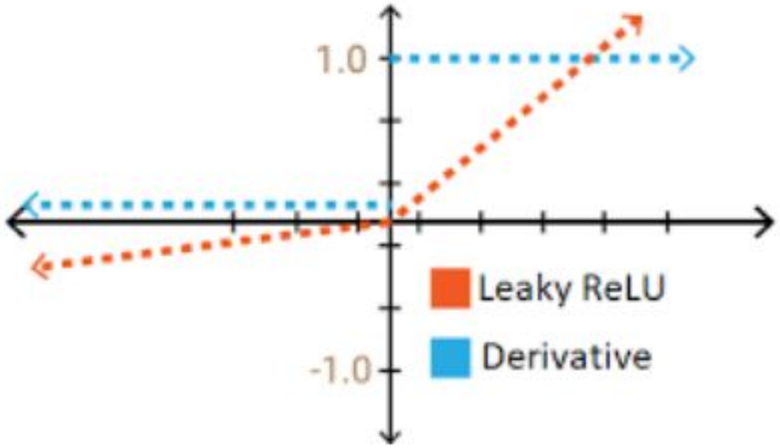
Observations:

- Greatly increase training speed compared to tanh and sigmoid
- Reduces likelihood of killing(vanishing) gradient
- It can blow up activation
- Dead nodes

Leaky-ReLU

$$f(x) = \max(0.01x, x)$$

$$\frac{df(x)}{dx} = \begin{cases} 0.01, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



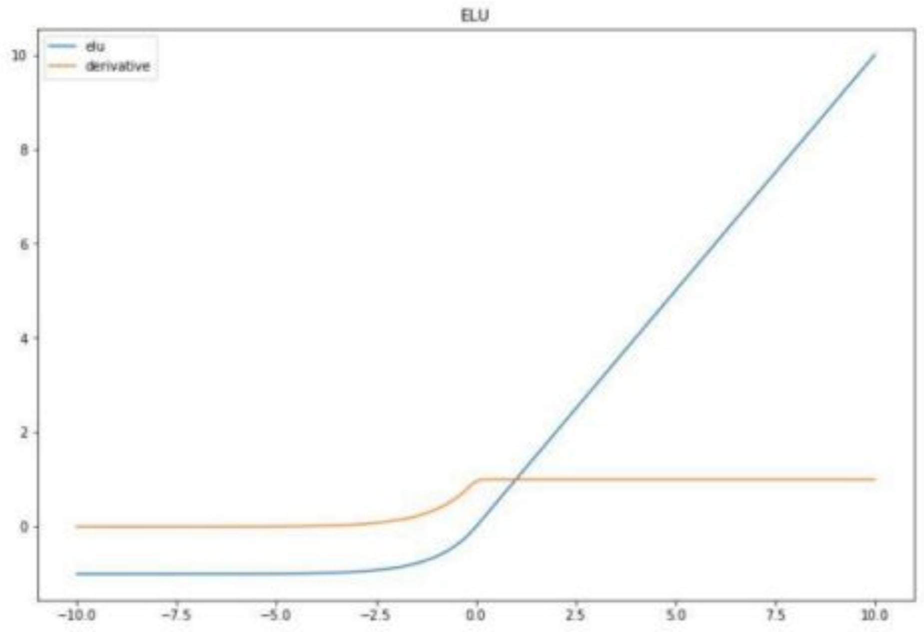
Observations:

- Fixed dying ReLU

Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} \alpha(e^x - 1), & x < 0 \\ 1x & x \geq 0 \end{cases}$$

$$\frac{df(x)}{dx} = \begin{cases} f(x) + \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases}$$



Observations:

- **It can produce –ve output**
- **It can blow up activation function**

Deep Learning Frameworks and Tools

Keras

TensorFlow

PyTorch

Popularity of PyTorch, TensorFlow, and Keras

Deep Learning and Generative AI

Features	TensorFlow	PyTorch	Keras
Written In	C++, CUDA, Python	Lua	Python
Architecture	Not easy to use	Complex, less readable	Simple, concise, readable
API Level	High and Low	Low	High
Datasets	Large datasets, high-performance	Large datasets, high-performance	Smaller datasets
Debugging	Difficult to conduct debugging	Good debugging capabilities	Simple network, so debugging is not often needed
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Second most popular	Third most popular	Most Popular
Speed	Fast, high-performance	Fast, high-performance	Slow, low performance