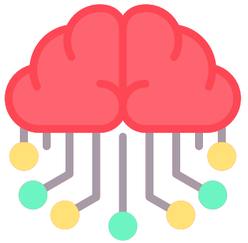
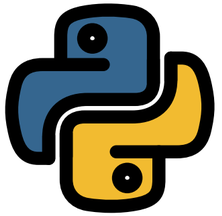
AISCIENCES
WHERE LEARNING BEGINS


Deep Learning with PYTORCH

INSTRUCTOR: ZIA AHMAD

WWW.AISCIENCES.IO

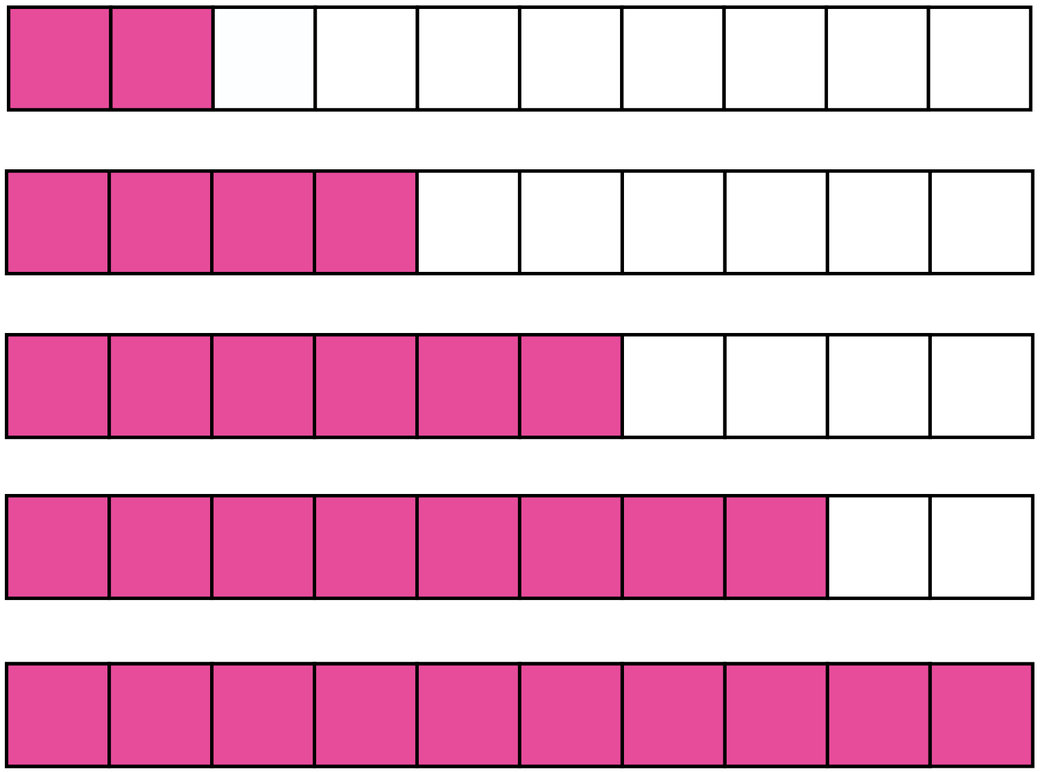
INSTALLING PYTORCH

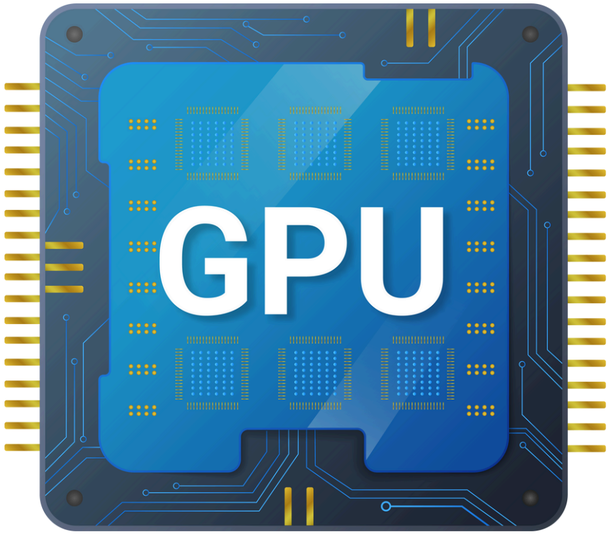
DEEP LEARNING

BASICS OF PYTORCH

DEEP LEARNING

LIST VS ARRAY VS TENSOR



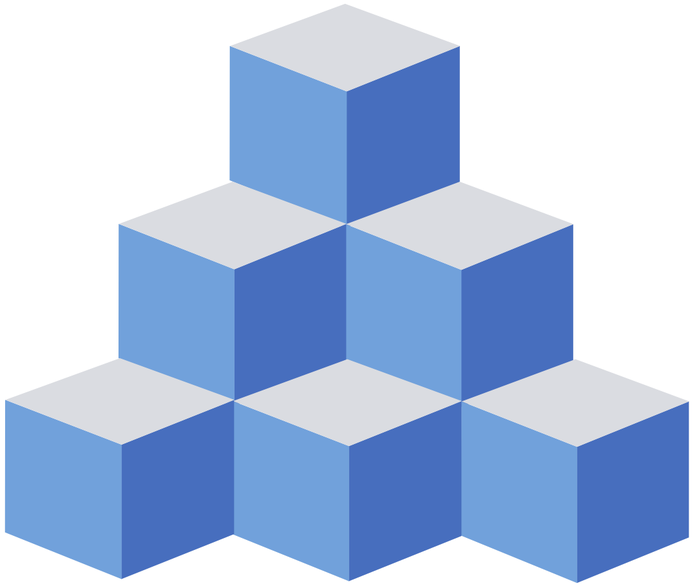


	List	Array	Tensor
Type	Heterogeneous (elements can be of different types).	Homogeneous (elements are of the same type).	Homogeneous (elements are of the same type).
Dimensionality	Typically one-dimensional, can contain nested lists for higher dimensions.	Typically one-dimensional or multi-dimensional (2D, 3D, etc.).	Multi-dimensional, can have any number of dimensions (scalars, vectors, matrices, and higher).
Mutability	Mutable (elements can be changed).	Mutable (elements can be changed).	Mutable (elements can be changed).
Indexing	Zero-based indexing.	Zero-based indexing, often supports multi-dimensional indexing.	Zero-based indexing, supports multi-dimensional indexing.

	List	Array	Tensor
Performance	Generally slower for numerical operations due to heterogeneity.	Faster for numerical operations due to homogeneity and optimized implementations.	Optimized for numerical operations, especially with hardware acceleration (e.g., GPUs).
Libraries	Built-in Python data structure.	Provided by libraries like NumPy.	Provided by libraries like TensorFlow, **PyTorch**.
Use Cases	General-purpose data storage, especially for mixed data types.	Efficient numerical computing, handling arrays, and matrices.	Machine learning, deep learning, and complex numerical computations involving high-dimensional data.
AutoGrad	Not Available	Not Available	**Available**

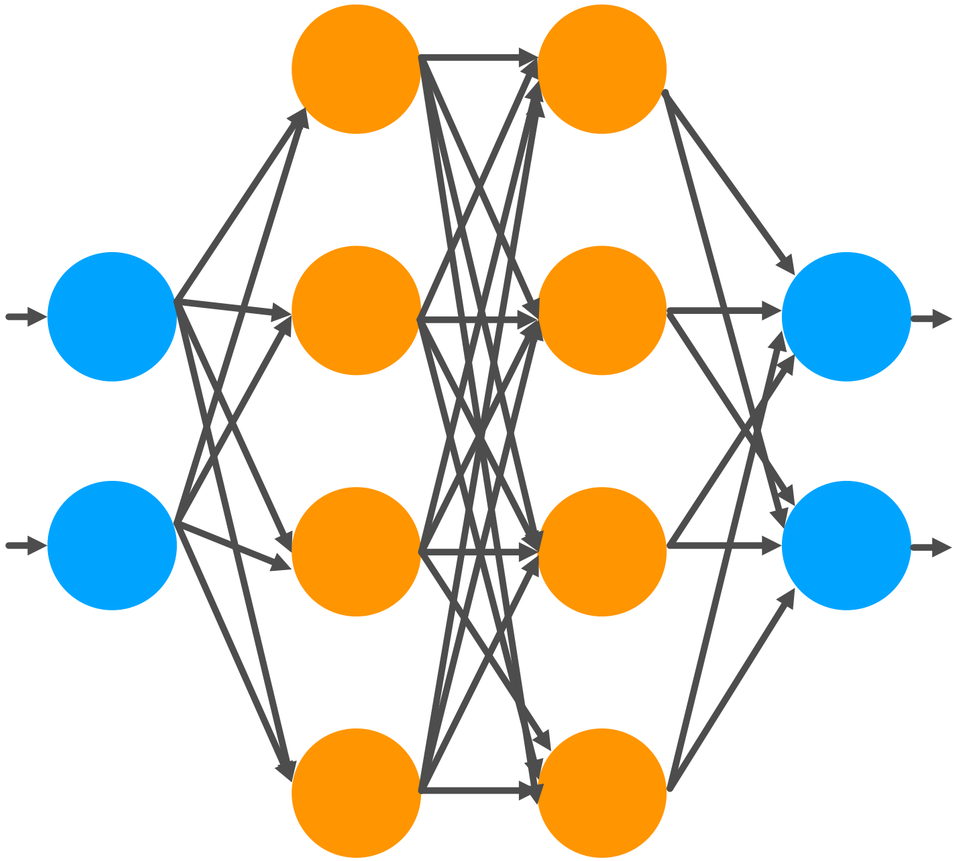
PyTorch

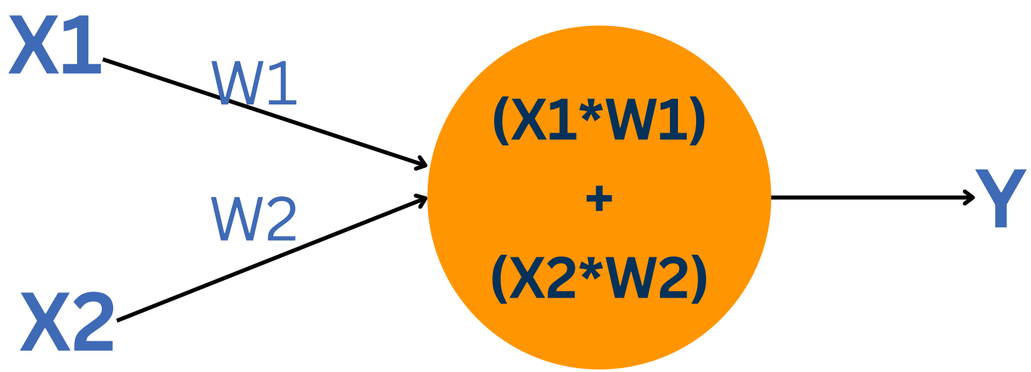
TENSORS



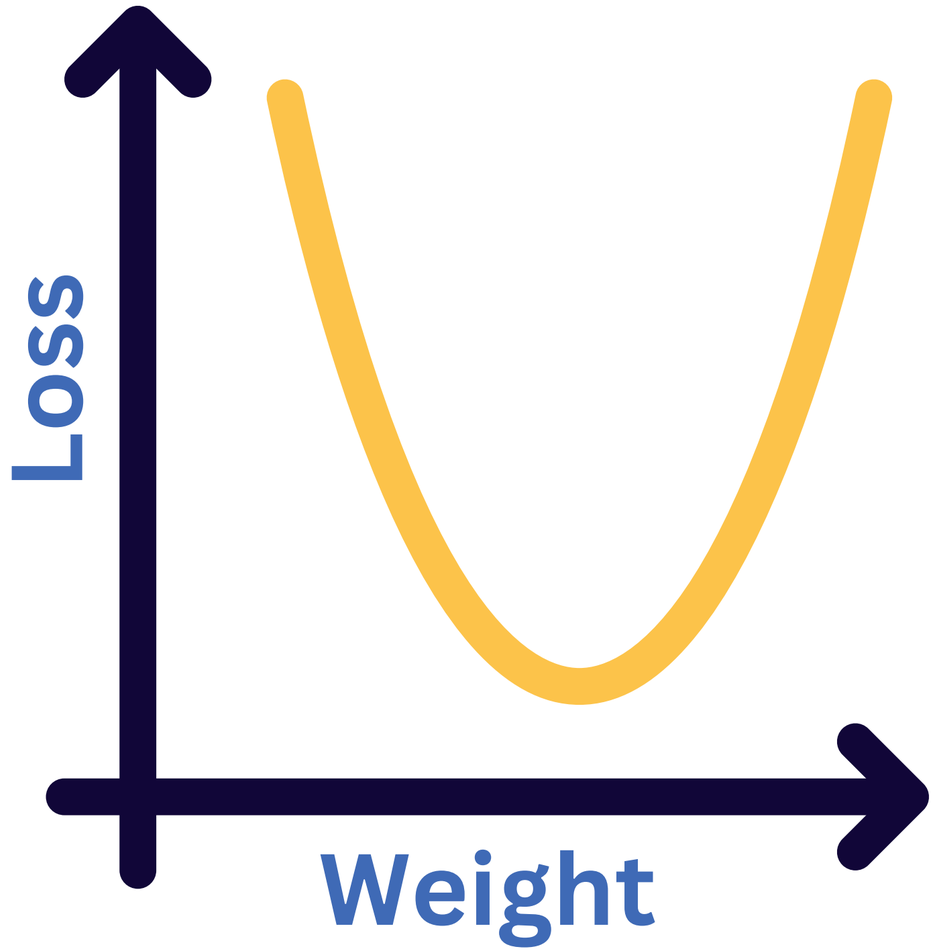
WWW.AISCIENCES.IO

AUTO-GRAD



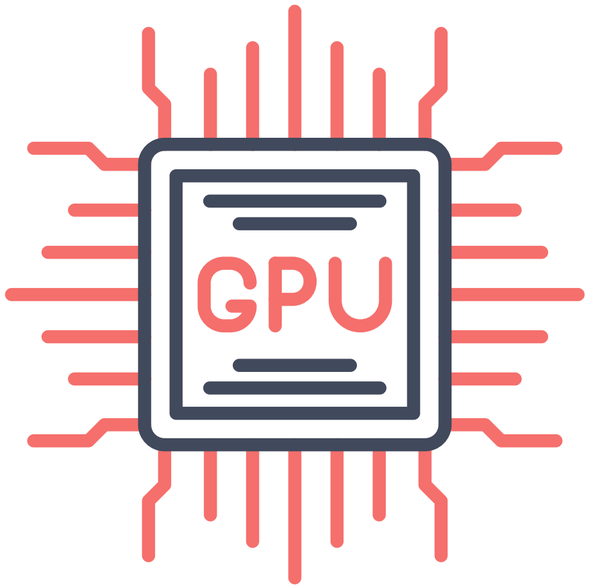


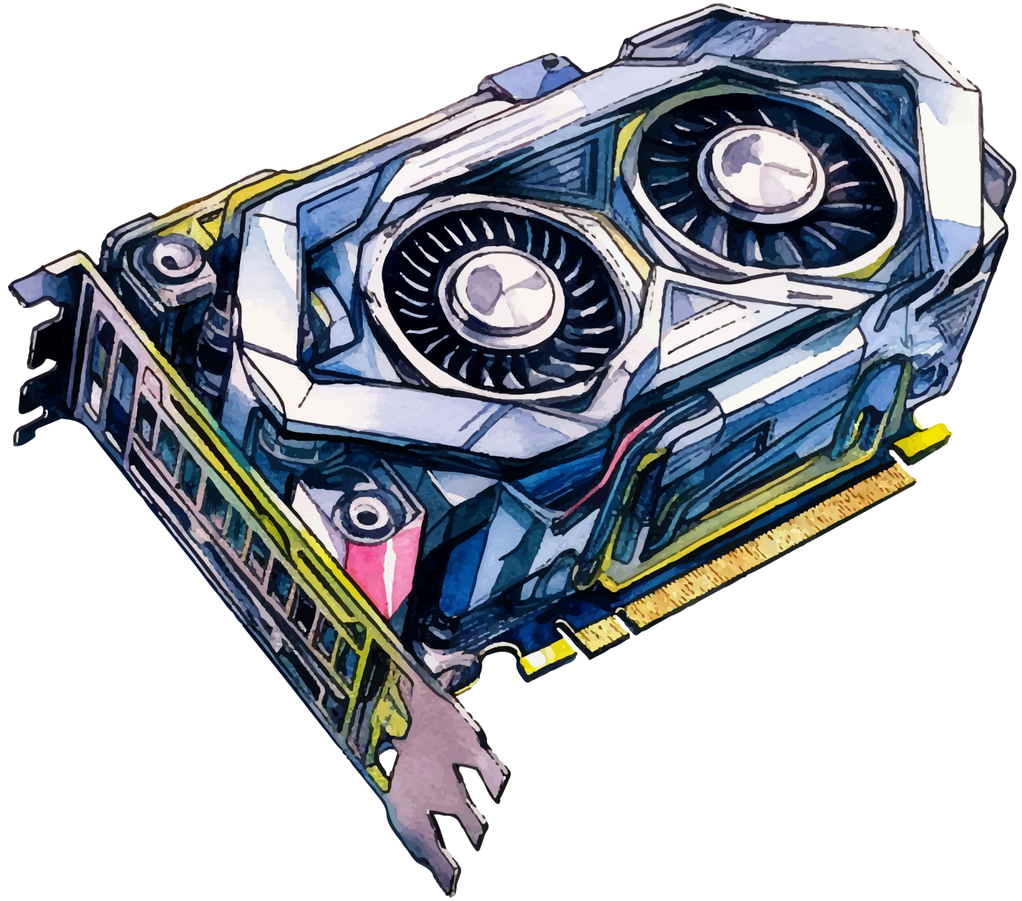
GRADIENT DECENT



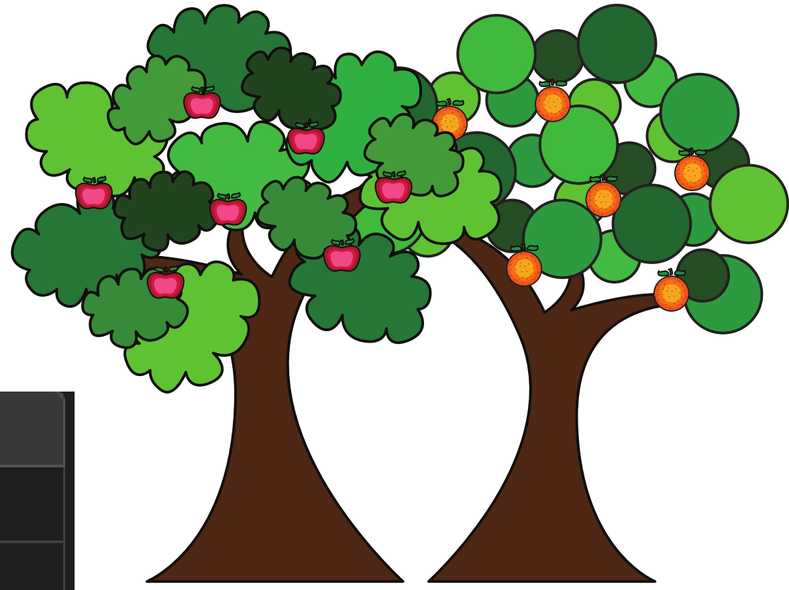
$$\frac{dy_{(\text{loss})}}{dx_{(\text{weight})}}$$

GPU



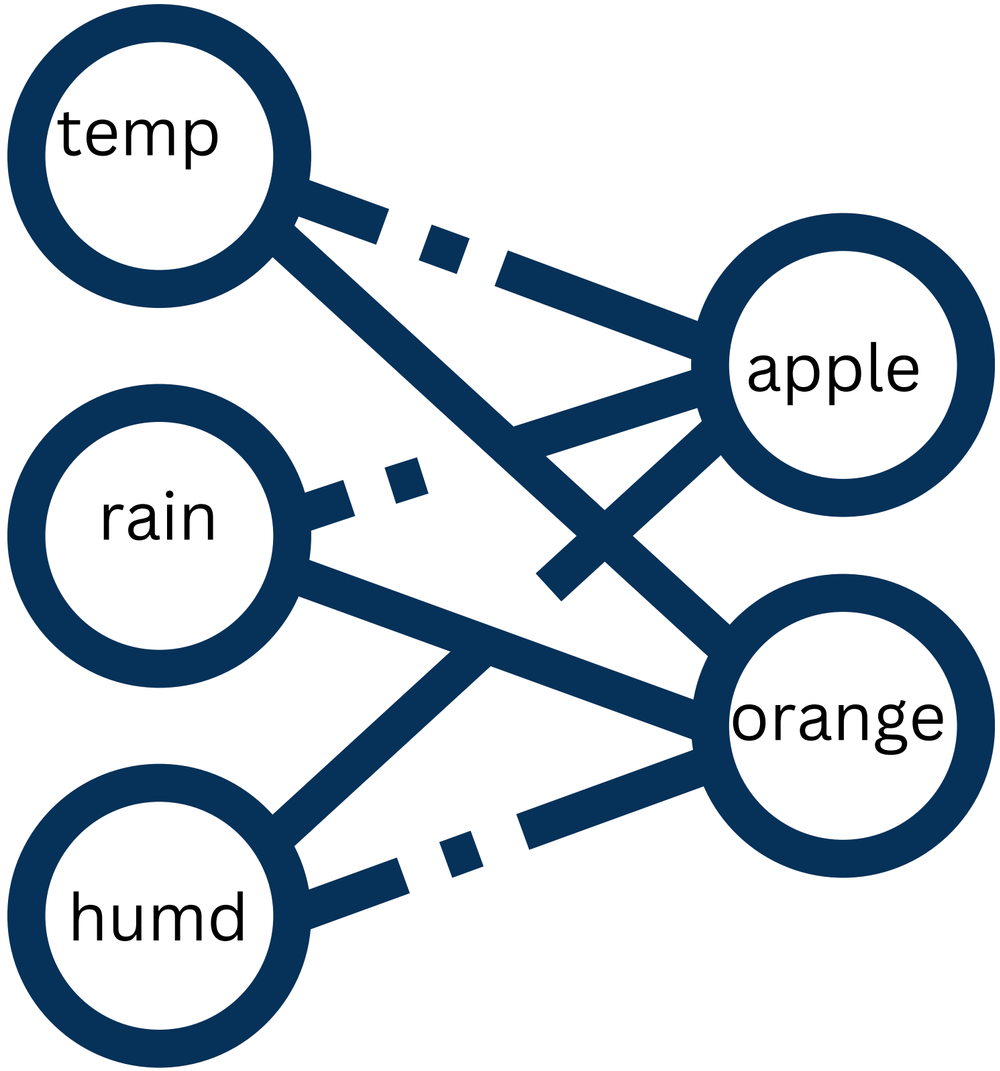


DATASET



Index	Temperature	Rainfall	Humidity	Apples	Oranges	Cities
0	73	67	43	56	70	New York
1	91	88	64	81	101	Los Angeles
2	87	134	58	119	133	Chicago
3	102	43	37	22	37	Houston
4	69	96	70	103	119	Phoenix
5	85	100	60	98	110	Philadelphia
6	95	80	55	88	95	San Antonio
7	105	120	75	115	140	San Diego
8	78	90	50	76	85	Dallas
9	82	70	45	65	75	Austin

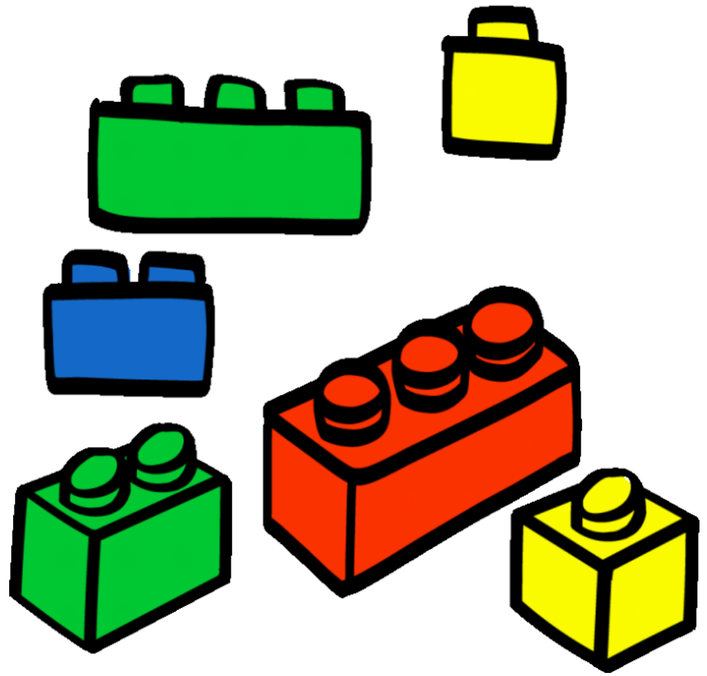
SIMPLE NEURAL NETWORK



$$X * W + b$$

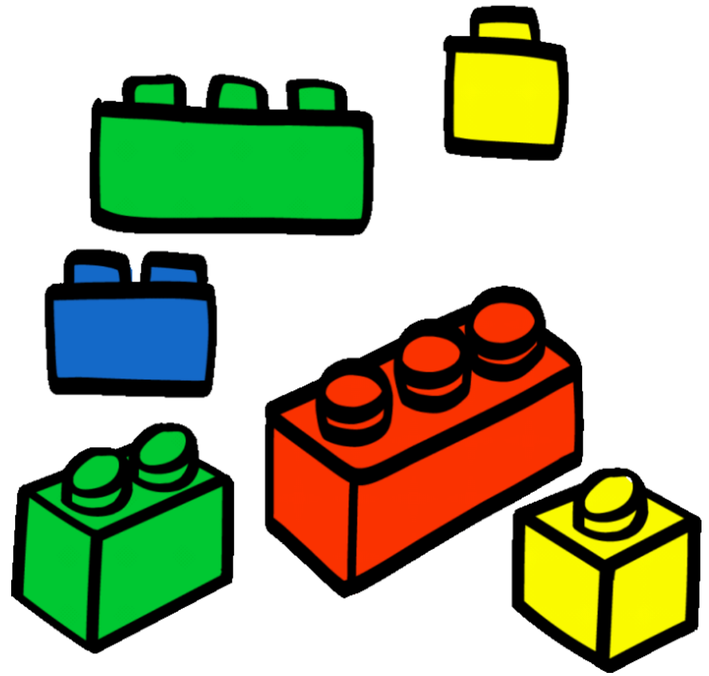
IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function
- Optimizer
- Activation Function



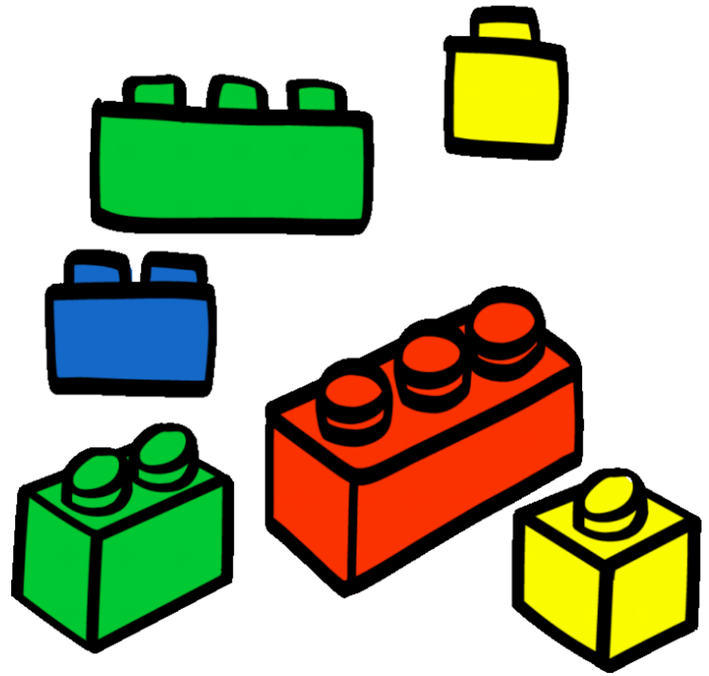
IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function
- Optimizer
- Activation Function

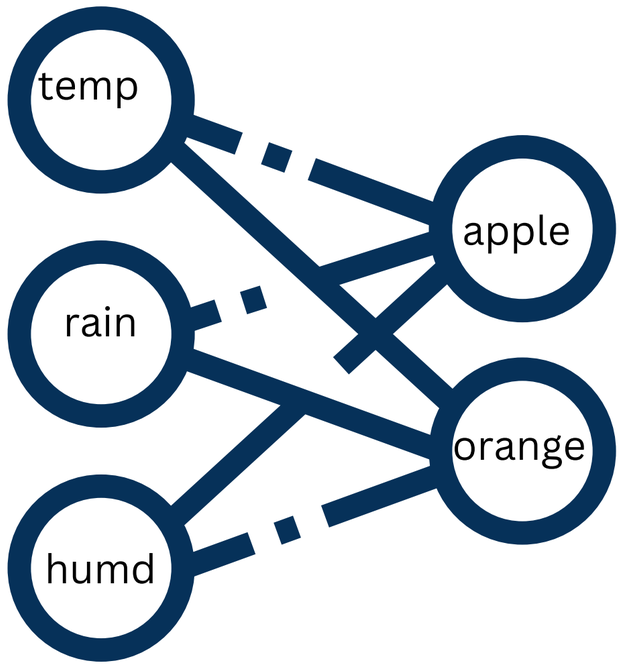


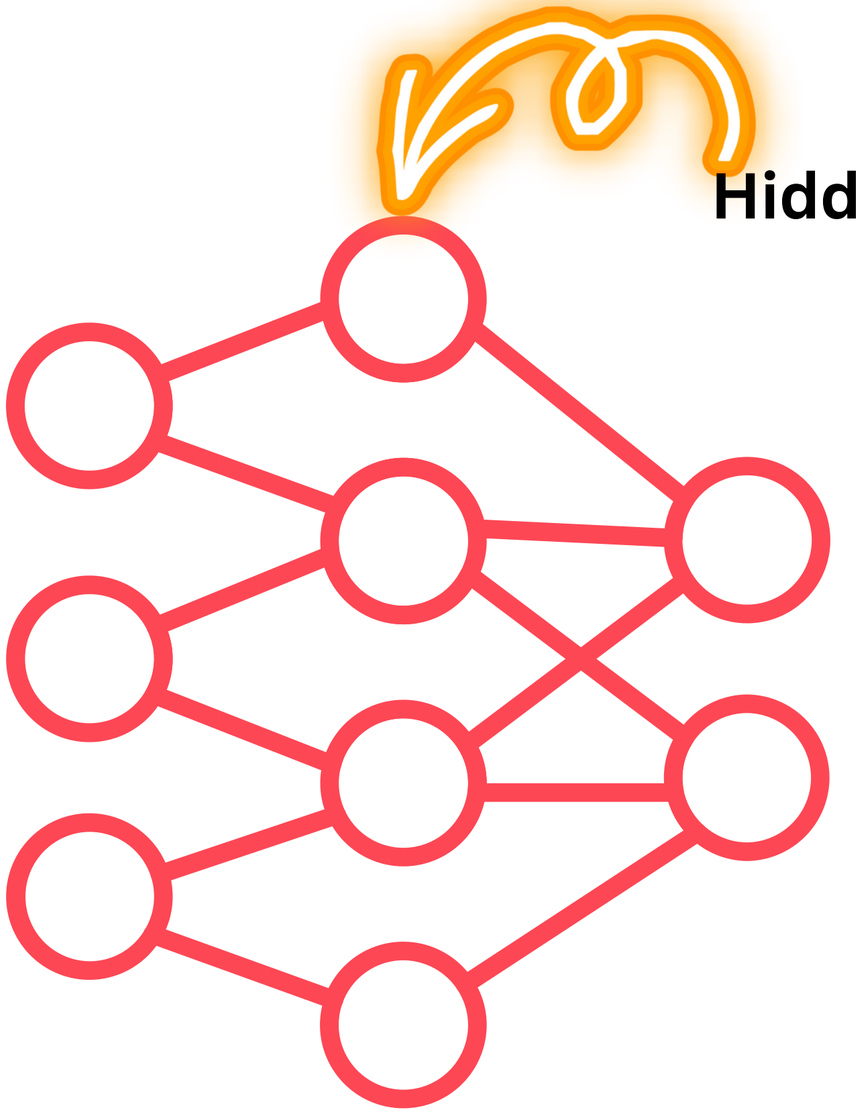
IMPORTANT ELEMENTS OF NEURAL NETWORK

- Loss Function ✓
- Optimizer
- Activation Function ✓



~~SIMPLE~~ NEURAL NETWORK



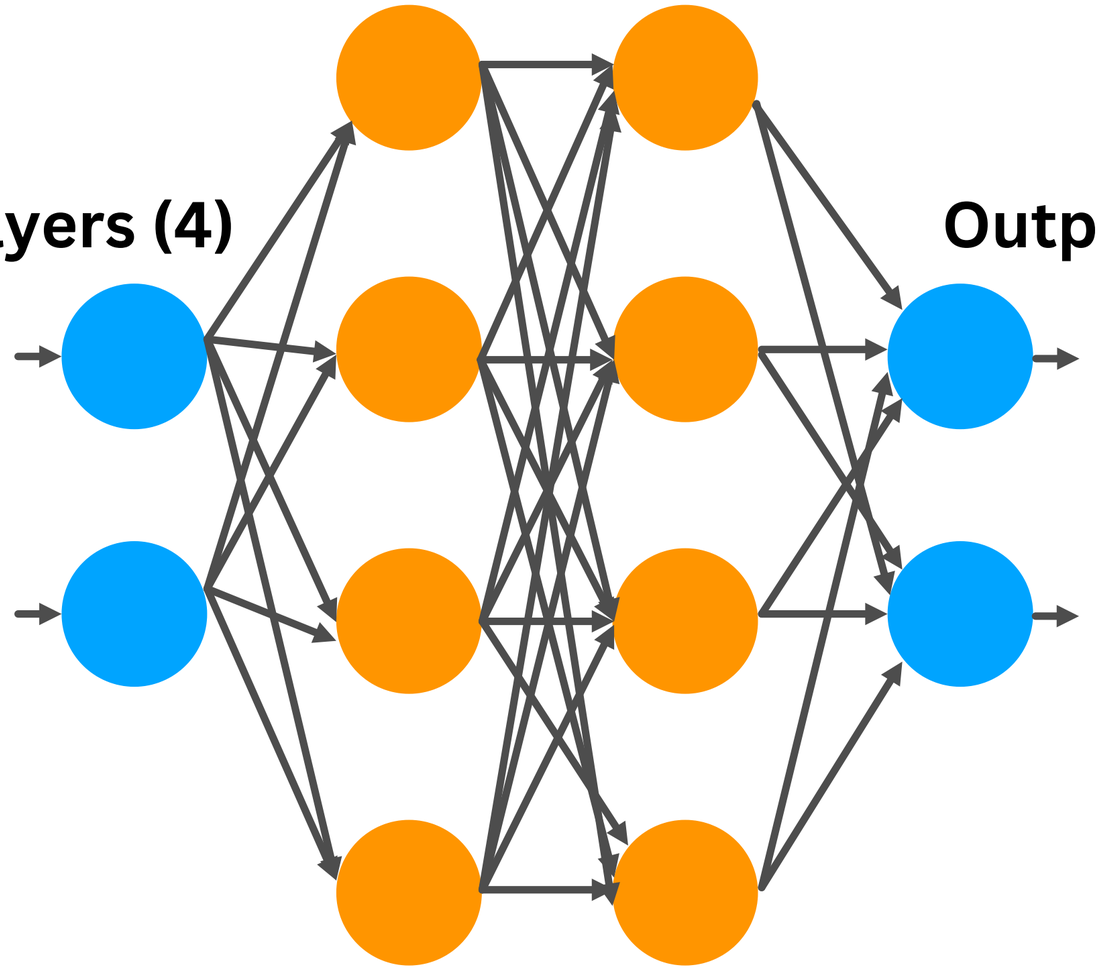


DEEP NEURAL NETWORK

Hidden Layers (6,4)

Input Layers (4)

- Sepal Length
- Sepal Width
- Petal Length
- Petal Width

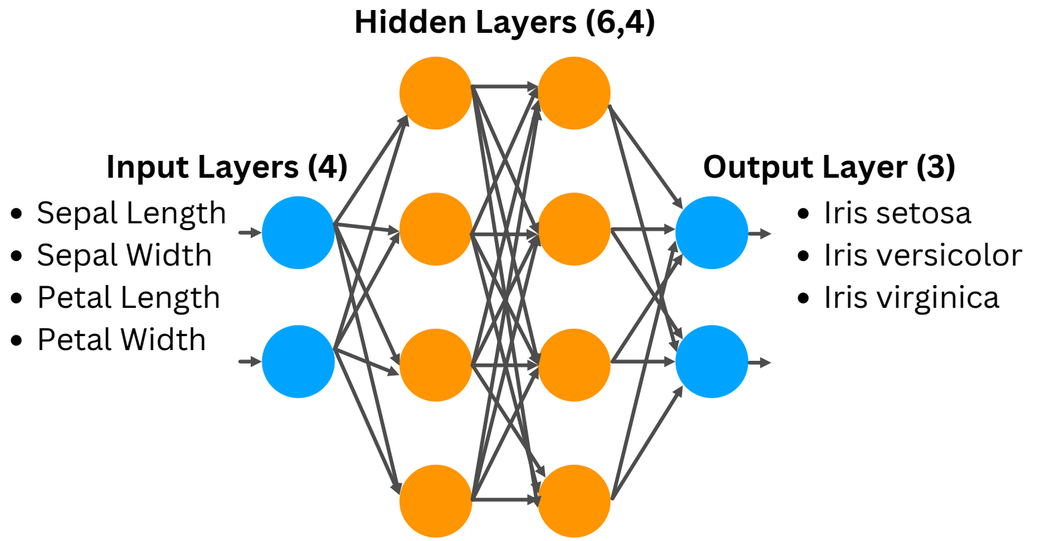


Output Layer (3)

- Iris setosa
- Iris versicolor
- Iris virginica

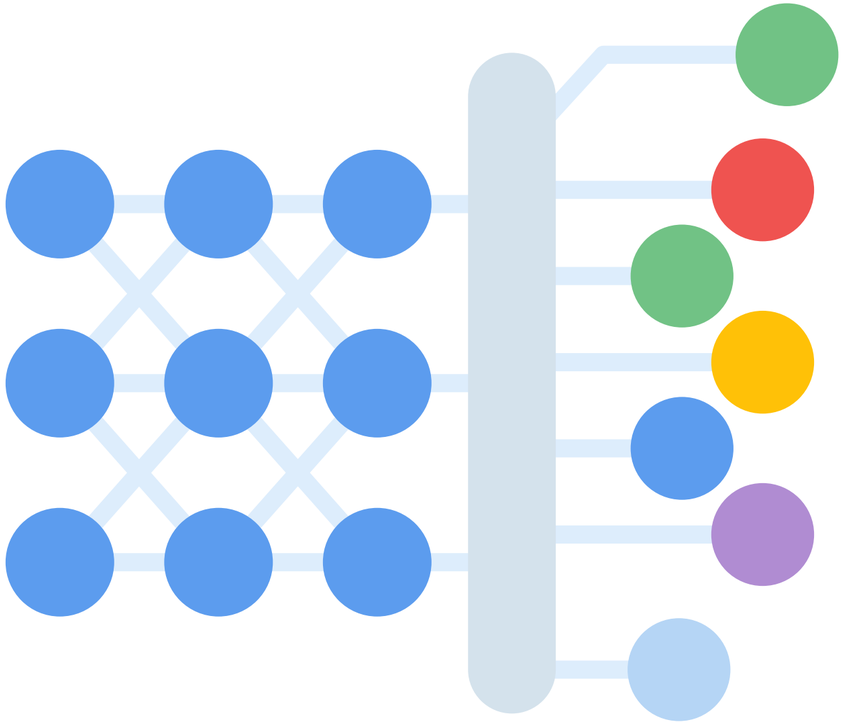
GOALS

- Create Tensors
- Train-Test Split (80-20)
- Create:
  - Model
  - Loss Function
  - Activation Function
  - Optimizer
- Report Error as Graph
- Output Predictions on Testing set



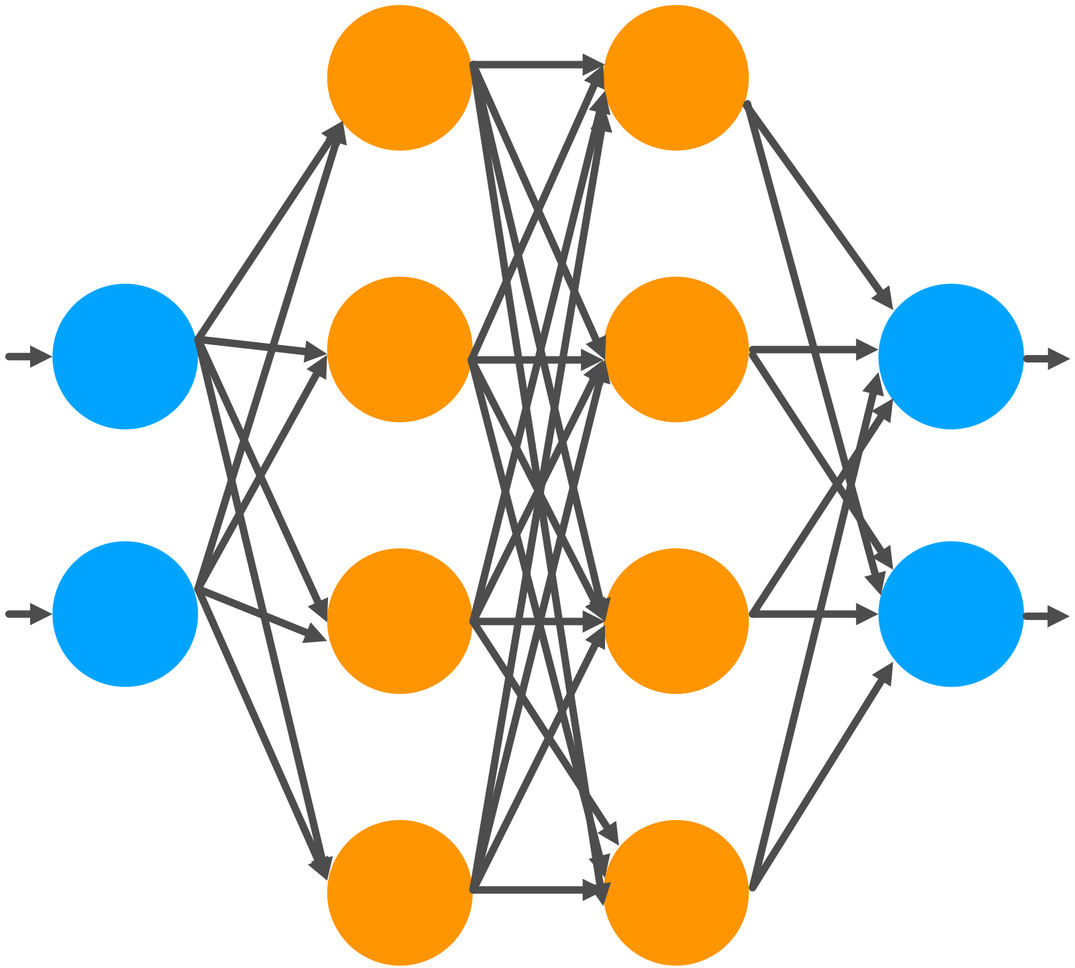
PyTorch

CNN



WWW.AISCIENCES.IO

ANN



CONVOLUTION

Input

$$\begin{bmatrix} 1 & 2 & 0 & 1 & 3 \\ 4 & 1 & 0 & 2 & 1 \\ 2 & 3 & 1 & 0 & 1 \\ 1 & 2 & 0 & 1 & 3 \\ 3 & 1 & 2 & 3 & 0 \end{bmatrix}$$

Kernal

$$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$$

Output

$$\begin{bmatrix} 6 & 3 & 2 \\ 5 & 2 & 1 \\ 4 & 1 & 0 \end{bmatrix}$$

Output(1,2) = (2×1)+(0×0)+(1×−1)+(1×1)+(0×0)+(2×−1)+(3×1)+(1×0)+(0×−1)=2+0−1+1+0−2+3+0−0=3

MAX POOLING

Input

$$\begin{bmatrix} 6 & 3 & 2 \\ 5 & 2 & 1 \\ 4 & 1 & 0 \end{bmatrix}$$

Pooling matrix 2x2

Output

$$\begin{bmatrix} 6 & 2 \\ 4 & 0 \end{bmatrix}$$

PROJECT

Brain Tumor Detection

- Import dataset
- Transform dataset
- Create CNN
  - Create Convolution Layer
  - Create Pooling Layers
  - Create Fully connected Layers
  - Write Forward Propagation Function
- Write Function for training
  - Select:
    - loss function
    - Optimizer
    - Activation function
  - Report error