Objective: Build a neural network for hand-written number detection.

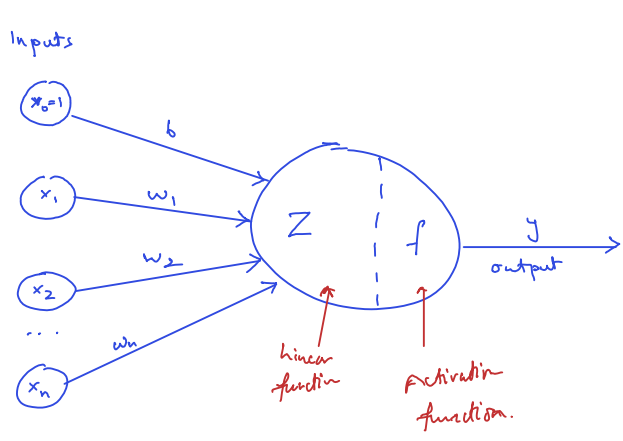
# Theory -

## Definition of a neural network

They are machine learning models that mimic the complex functions of the human brain. These models consist of interconnected nodes or neurons that process data, learn patterns, and enable tasks such as pattern recognition & decision-making.

It follows a three-stage process:

1. Input Computation: Data is fed into the network.
2. Output Generation: Based on the current parameters, the network generates an output.
3. Iterative Refinement: The network refines its output by adjusting weights & biases, gradually improving its performance on diverse tasks.




## Working of a neural networks

### Forward Propagation

i/p layer → hidden layer → output layer

This process is known as forward propagation.

During this phase:

1. Linear Transformation: Each neuron in a layer recieves inputs, which are multiplied by the weights associated w/ the connections. These products are summed together, & a bias is added to the sum.

   Representation: $z = w_1x_1 + w_2x_2 + \dots + w_nx_n + b$

   w = weight,
   x = input value,
   b = bias.

2. Activation: The result of linear transformation (z) is then passed through an activation fn. The activation fn. is crucial because it introduces non-linearity into the system, enabling the network to learn more complex patterns.

   Ex: ReLU, sigmoid, and tanh

### Backward Propagation

After forward propagation, the network evaluates its performance using a loss fn, which measures the difference b/w the actual output & the predicted output. The goal of training is to minimize this loss. This is where backward propagation comes to play:

1. Loss Calculation: The network calculates the loss, which provides a measure of error in the predictions. The loss fn. could vary: common choices are mean squared error for regression tasks or cross-entropy loss for calculation.

2. Gradient Calculation: The network computes the gradients of the loss fn. w/ respect to each weight & bias in the network. This involves applying the chain rule of calculus to find out how much each part of the o/p error can be attributed to each weight & bias.

3. Weight Update: Once the gradients are calculated, the weights & biases are updated using an optimization algorithm such as Stochastic Gradient Descent (SGD). The weights are adjusted in the opposite direction of the gradient to minimize the loss. The size of the step taken in each update is determined by the learning rate.

A neural network can be supervised or unsupervised,

In supervised learning, labeled data is used to train a neural network so that it may learn to map inputs to matching outputs.

Unsupervised learning works w/ unlabeled data & looks for structures or patterns in the data.

## Tools to be Used

- Python,
- numpy, — for numerical computation.
- matplotlib, — for visualizing the data.
- MNIST dataset. — ready to use labelled data
- pandas. — for reading & loading the data from the MNIST dataset.

# Steps for the Project

① Load & Preprocess the data.

MNIST dataset,
28 x 28 pixels in size (grayscale)

We use pandas to load & process.

② Defining the Neural Network Architecture

1 input layer, 2 hidden layer, 1 output layer

- 784 neurons ↓ 28 x 28
- 64 neurons & ReLU activation fn.
- 10 neurons & softMax activation fn.

③ Compute the gradients.

Repeat weight adjustment until algorithm converges on a set of weights that produces the lowest possible error.

Error % ≠ 0, this means the model has stopped learning.

④ Create a neural network object.

Initialize a neural network w/ 784 input features, 64 hidden nodes & 10 output neurons. And train the back propagation algorithm for 10 epochs. w/ a learning rate of 0.1.

⑤ Train the neural network.

Once the neural network object is created, it can now be trained using the dataset & then used to make predictions on new data.

⑥ Evaluating the neural network.

The performance of the network will be evaluated by comparing the predicted o/p w/ the actual o/p & calculating a metric such as accuracy.

Additionally, analyze any patterns or trends in the errors made by the network to identify areas for improvement.

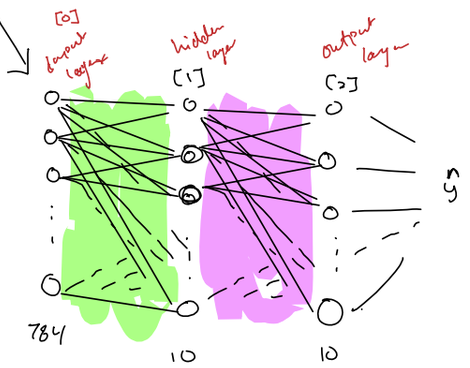
784 (28×28 px) — m training images → can be represented as matrices

$$X = \begin{bmatrix} — & x^{(1)} & — \\ — & x^{(2)} & — \\ & \vdots & \\ — & x^{(m)} & — \end{bmatrix}^T = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

(T) Transpose for calc. purposes

each row is 784 col's wide.
each col represents a pixel from the image
the value stored in each cell ∈ [0, 255]
color range
ex: 0 = white, 255 = black

now each column is an example.




## Forward Propagation Calc.

Basically taking an image from matrix and running it thru the network.

$A^{[0]} \in$ input layer $= X$ $(784 \times m)$ — X is any input from

Dealing w/ matrices

$$Z^{[1]}_{10\times m} = \underbrace{W^{[1]}}_{10\times 784} \cdot \underbrace{A^{[0]}}_{784\times m} + \underbrace{b^{[1]}}_{10\times 1 \Rightarrow 10\times m}$$

$W^{[1]}$ = weights matrix for hidden layer

$b^{[1]}$ = bias values matrix for hidden layer

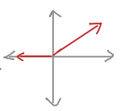
$Z^{[1]}$ unactivated hidden layer

Now, we apply an activation function to hidden layer matrix. Why? → The hidden layer is a linear combination of the input layer, & the output layer is a linear combination of the hidden layer. w/o the activation fn, there is no change so it is as it the o/p layer is the linear combination of the i/p layer. Obviously this is not desired. Also makes the model more complex & powerful.

Here, we use Rectified Linear Unit. (ReLU)

$$ReLU(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \le 0 \end{cases}$$

very simple fn. makes our graph look like



So,

$$A^{[1]} = g(Z^{[1]}) = ReLU(Z^{[1]})$$

Now, our unactivated second layer.

$$\underset{10\times m}{Z^{[2]}} = \underset{10\times 10}{W^{[2]}} \underset{10\times m}{A^{[1]}} + \underset{10\times 1 \Rightarrow 10\times m}{b^{[2]}}$$

This time we apply another activating fn.

$$A^{[2]} = softmax(Z^{[2]})$$

softmax(x) ← why? Since now we deal w/ the o/p layer, where each of the 10 nodes correspond to a number that represents the predicted output, we want each of them to have a probability a value ∈ (0, 1). So, to get that we use softmax.

now we get a prediction based on the image input :)

Example of softmax act. fn:

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_i}}$$

softmax always delivers in a range ∈ (0, 1)

o/p layer → softmax(x)ᵢ → Probabilities

| o/p layer | Probabilities |
|---|---|
| 1.3 | 0.02 |
| 5.1 | 0.90 |
| 2.2 | 0.05 |
| 0.7 | 0.01 |
| 1.1 | 0.02 |

$$\text{softmax}(1.3) = \frac{e^{1.3}}{e^{1.3}+e^{5.1}+e^{2.2}+e^{0.7}+e^{1.1}} = \frac{3.6693}{181.73413} = 0.02019\ldots \approx 0.02$$

## Backward Propagation

We use this algorithm to optimize the weights & biases

$dZ^{[2]}$ represents error of second layer

$$\underset{10\times m}{dZ^{[2]}} = \underset{10\times m}{A^{[2]}} - \underset{10\times m}{y}$$ → one-hot encode Ex, if y = 4 ⇒

$$\underset{10\times 10}{dW^{[2]}} = \frac{1}{m} \underset{10\times m}{dZ^{[2]}} \underset{m\times 10}{A^{[1]T}}$$

aka. derivative of Loss fn. — to calculate how much the weights contributed to output.

$$\underset{10\times 1}{db^{[2]}} = \frac{1}{m} \underbrace{\sum \underset{10\times 1}{dZ^{[2]}}}_{\text{to calculate bias contribution.}}$$ — average formula, Avg. of absolute error

for 2nd layer

for 1st layer

$$\underset{10\times m}{dZ^{[1]}} = \underbrace{\underset{10\times 10}{W^{[2]T}} \underset{10\times m}{dZ^{[2]}}}_{\text{basically applying the weights in reverse to retrace layer to get weights of first layer}} * \underbrace{g'(\underset{10\times m}{Z^{[2]}})}_{\text{derivative of activation fn}}$$

$$\underset{10\times 784}{dW^{[1]}} = \frac{1}{m} \underset{10\times m}{dZ^{[1]}} \underset{m\times 784}{X^T}$$

$$\underset{10\times 1}{db^{[1]}} = \frac{1}{m} \sum \underset{10\times 1}{dZ^{[1]}}$$

Updating weights

$$W^{[1]} = W^{[1]} - \alpha\, dW^{[1]}$$
$$b^{[1]} = b^{[1]} - \alpha\, db^{[1]}$$
$$W^{[2]} = W^{[2]} - \alpha\, dW^{[2]}$$
$$b^{[2]} = W^{[2]} - \alpha\, db^{[2]}$$

$\alpha$ = learning rate

$\alpha$ is a hyperparameter, it's set by us.

Repetition for set 'epochs'.