6CS012 – Artificial Intelligence and Machine Learning.
Tutorial – 04
Starting with Deep Neural Network.
Training the Fully Connected Deep Neural Network.

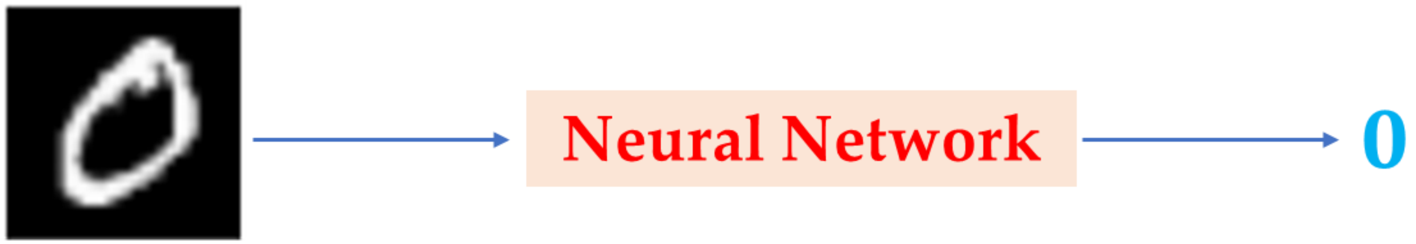
Siman Giri {Module Leader – 6CS012}

1. Deep Neural Network: Essential Components.

{Revisiting key Ideas from Lecture.}

1.1 Let’s First Define a Task.

Build a Fully Connected Neural Network to Classify English Handwritten Digits.



1.2 Needs to have a Data.

- For this task we will be using **MNIST dataset consisting of handwritten images**.
- Few feature of the dataset:
  - Consist of **60,000** gray scale example image for the **training set**.
  - Consist of **10,000** gray scale example image for the **test set**.
  - All the images are of shape $\mathbf{28 \times 28}$.
  - Each pixel of the image are represented by a value between **0 and 255** where:
    - 0 : represents black
    - 255: represents white
    - Anything in-between is different shades of grey.

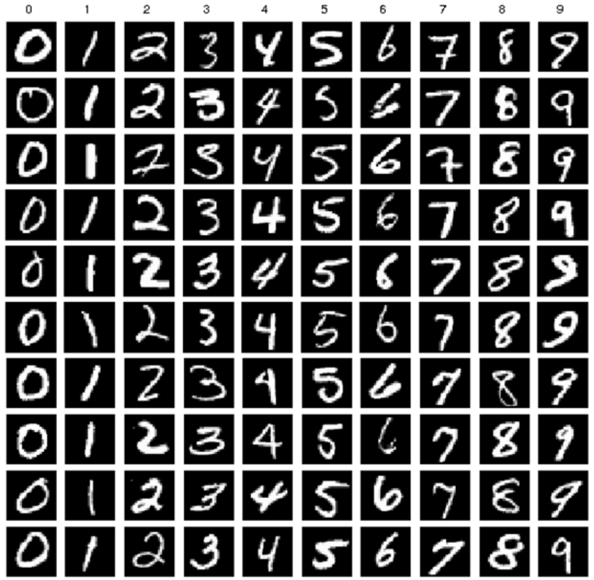


Fig: Collection of Handwritten Digits from MNIST Dataset.

1.2.1 Data has to be Pre – Processed.

- **Some common Data Preprocessing we do:**
  - If possible, **make sure all the images are of same shape this** will help you to design your architecture further down.
  - **Detect corrupted images:**
    - Use scripts to identify and remove unreadable images.
  - **Visualize samples:**
    - Plot a few images from each class to verify correctness.
  - **Normalization:**
    - Normalize pixel values to ensure consistent input for the model:
      - Convert pixel values to the range $[\mathbf{0}, \mathbf{1}]$ by dividing by $\mathbf{255}$.
      - Alternatively, use standardization: $\frac{(\textbf{image} - \textbf{mean})}{\textbf{std.Deviation}}$.

- **Cautions**: Always Normalize after proper train – Val and test split.
  - {Prevents Data Leakage and Helps in Generalizations.}

Correct Work Flow:
1. **Split the dataset** → Training, Validation, Test
2. **Compute mean and std from the training set only**
3. **Normalize training, validation, and test sets using the same parameters**

1.3 Defining a Model.

- For the Task we proposed to use **Multi Layer Fully Connected Neural Network**:
  - It is called Fully Connected Neural Network;
    - as in this design all the neurons are connected with each other.
  - Each individual neuron performs two task:
    - **Compute a weighted sum (applies Linear Transformation) and**
    - **Applies an activation function.**

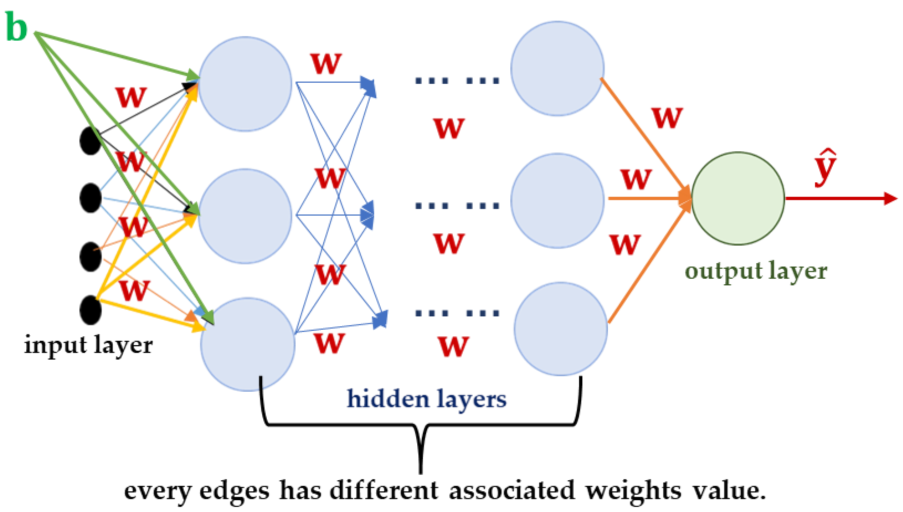
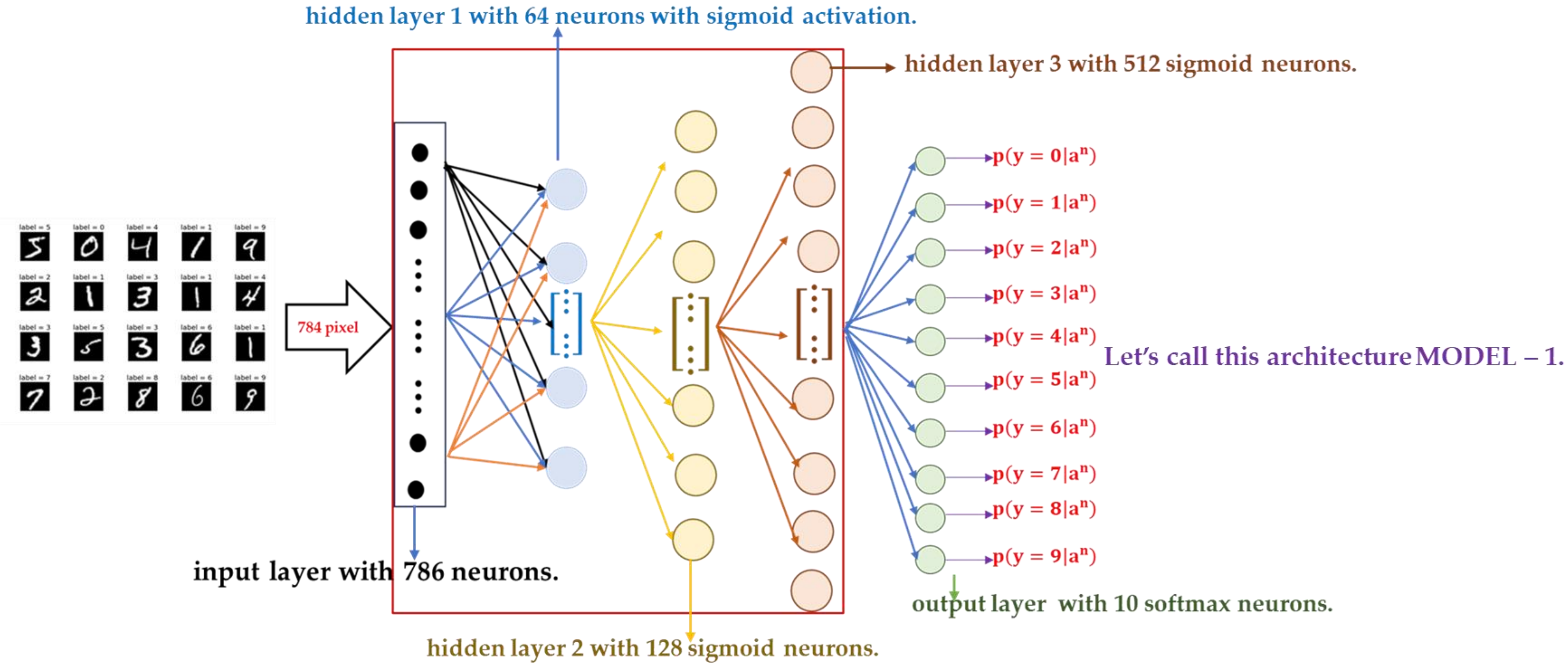
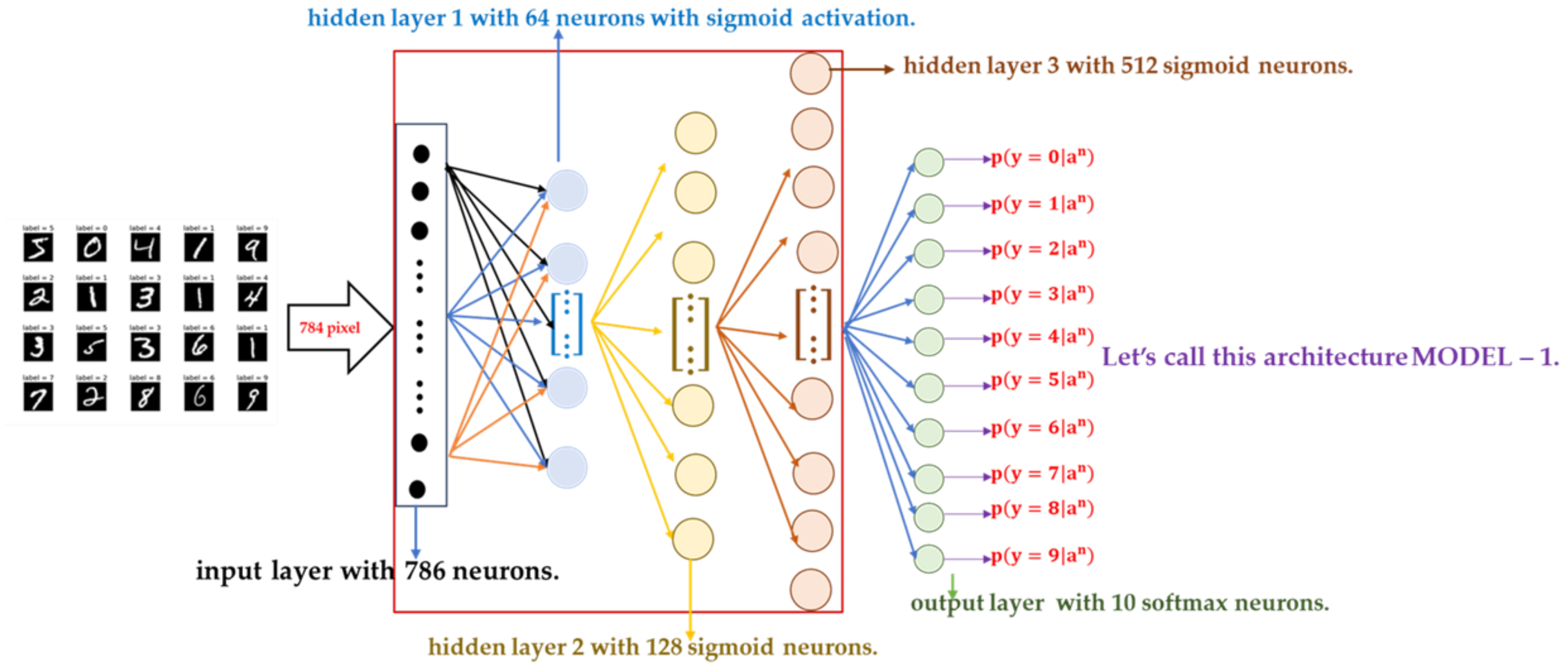


Fig: A deep network with n hidden layers.

1.3.1 Pick an Architecture.



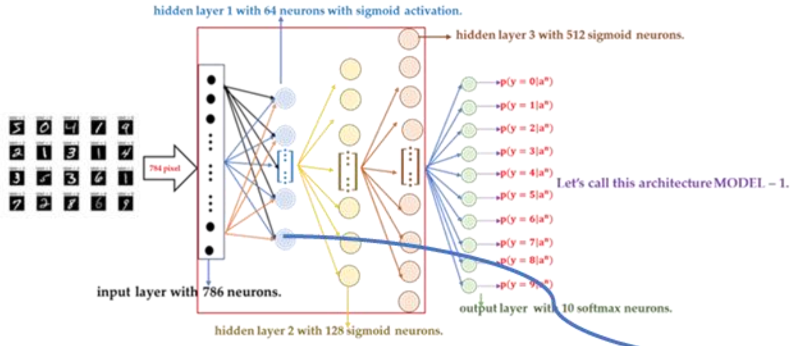
1.3.2 Explore the Architecture : Input Layer.



1. Input Layer:

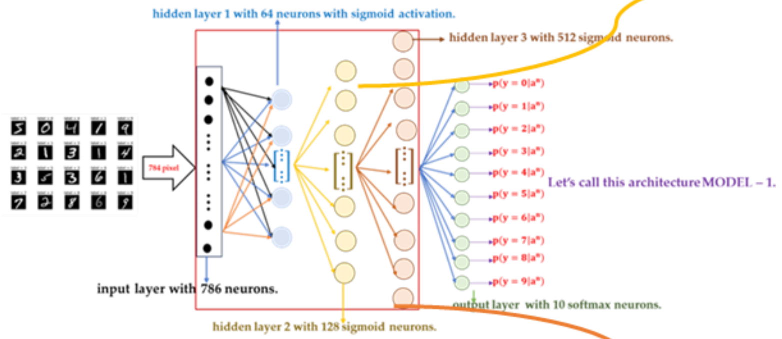
Input $\mathbf{x} \in \mathbb{R}^{784}$, i.e. $\mathbf{x} \rightarrow \begin{bmatrix} \text{pixel}_0 \\ \text{pixel1} \\ \vdots \\ \vdots \\ \text{pixel}_{784} \end{bmatrix}$ {flattened $28 \times 28 \times 1$ image}

1.3.2 Explore the Architecture : Hidden Layer - 1.



2. **Hidden Layer:**
   - For each hidden layer, the output is a transformation(weighted sum and activation) of the previous layer's output using the weight matrix and bias vector.
     - First Hidden Layer - 64 neurons.
     - The output $h^1 \in \mathbb{R}^{64}$ is computes as:
     - $\mathbf{h^1 = \sigma(z^1) \rightarrow \{z^1 = W^1x + b^1\}}$
   - here:
     - $W^1 \in \mathbb{R}^{64\times784}$ is a weight matrix for first hidden layer.
     - $$W^1 = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & \cdots & \cdots & w^1_{1,784} \\ w^1_{1,2} & \ddots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ w^1_{64,1} & w_{64,2} & \cdots & \cdots & \cdots & w_{64,784} \end{bmatrix}; x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ \vdots \\ \vdots \\ x_{784} \end{bmatrix}; b^1 = \begin{bmatrix} b^1_1 \\ b^1_2 \\ \vdots \\ \vdots \\ \vdots \\ b^1_{64} \end{bmatrix}$$
     - $z^1 = \begin{bmatrix} z^1_1 \\ \vdots \\ \vdots \\ z^1_{64} \end{bmatrix}$; each element computes as: $z^1_i = \sum_{j=1}^{784} w_{ij} + b_i$,
     - **In Matrix form:**
       - $$z^1 = \begin{bmatrix} w^1_{1,1}x_1 + w^1_{1,2}x_2 + \cdots + w^1_{1,784}x_{784} + b^1_1 \\ w^1_{2,1}x_1 + w^1_{2,2}x_2 + \cdots + w^1_{2,784}x_{784} + b^1_2 \\ \vdots \\ w^1_{64,1}x_1 + w^1_{64,2}x_2 + \cdots + w^1_{64,784}x_{784} + b^1_{64} \end{bmatrix}$$
     - $$h^1 = \sigma\left(\begin{bmatrix} z^1_1 \\ \vdots \\ \vdots \\ z^1_{64} \end{bmatrix}\right) \Rightarrow \begin{bmatrix} \sigma(z^1_1) \\ \vdots \\ \vdots \\ \sigma(z^1_{64}) \end{bmatrix} \{\text{element wise sigmoid activation function.}\}$$

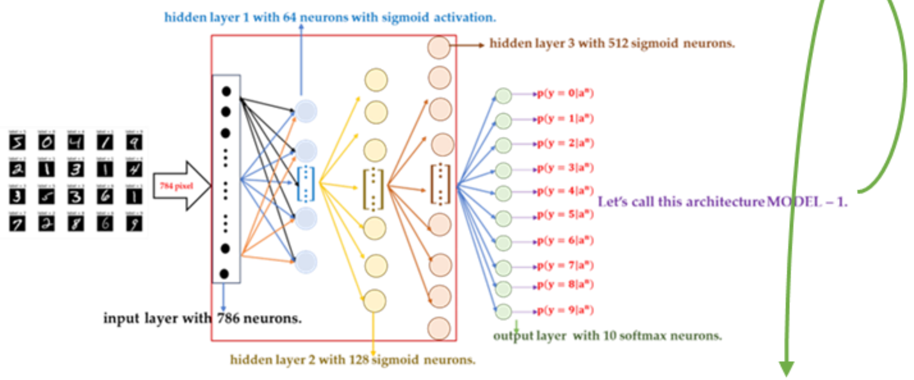
1.3.2 Explore the Architecture : Hidden Layer - 2.



2. **Hidden Layer:**
   - **Second Hidden Layer – 128 neurons:**
     - The output: $\mathbf{h}^2 = \boldsymbol{\sigma}(\mathbf{z}^2)\ \{\mathbf{z}^2 = \mathbf{W}^2\mathbf{h}^1 + \mathbf{b}^2\}$
       - Here:
         - $\mathbf{W}^2 \in \mathbb{R}^{128\times 64}$ is the **weight matrix** for the second hidden layer.
         - $\mathbf{b}^2 \in \mathbb{R}^{128}$ is the **bias vector** for the second hidden layer.
         - $\mathbf{h}^1 \in [\mathbf{0\ to\ 1}]^{64}$ is the **vector of activated output** from **hidden layer 1**.

2. **Hidden Layer:**
   - **Third Hidden Layer – 512 neurons:**
     - The output: $\mathbf{h}^3 = \boldsymbol{\sigma}(\mathbf{z}^3)\ \{\mathbf{z}^3 = \mathbf{W}^3\mathbf{h}^2 + \mathbf{b}^3\}$
       - Here:
         - $\mathbf{W}^3 \in \mathbb{R}^{512\times 128}$ is the **weight matrix** for the second hidden layer.
         - $\mathbf{b}^3 \in \mathbb{R}^{512}$ is the **bias vector** for the second hidden layer.
         - $\mathbf{h}^2 \in [\mathbf{0\ to\ 1}]^{128}$ is the **vector of activated output** from **hidden layer 2**.

1.3.2 Explore the Architecture: Output Layer.



3. Output Layer – 10 Neurons:

- The Output $\hat{\mathbf{y}} \in \mathbb{R}^{10}$ (the predicted class probabilities) is computed as:
  - $\hat{\mathbf{y}} = \mathbf{Softmax}(\mathbf{z}^4)\{\mathbf{z}^4 = \mathbf{W}^4\mathbf{h}^3 + \mathbf{b}^4\}$
- Here:
  - $\mathbf{W}^4 \in \mathbb{R}^{10\times 512}$ is the **weight matrix** for the **output layer**. 10 because there are 10 classes.
  - $\mathbf{b}^4 \in \mathbb{R}^{10}$ is the **bias vector** of the **output layer**.
  - The softmax function is applied element wise to the vector:
    - $\hat{\mathbf{y}}_{\mathbf{j}} = \dfrac{e^{(\mathbf{w}^4\mathbf{h}^3+\mathbf{b}^4)_j}}{\sum_{k=1}^{10} e^{(\mathbf{w}^4\mathbf{h}^3+\mathbf{b}^4)_k}}\ \forall \mathbf{j} \in \{\mathbf{1}, \mathbf{2}, \dots, \mathbf{10}\}$
  - The **predicted class** $\hat{\boldsymbol{y}}$ is the index of **the maximum value in** $\hat{\boldsymbol{y}}$:
    - $\hat{\mathbf{y}} = \mathbf{argmax}_j\ \hat{\mathbf{y}}_{\mathbf{j}}$.

1.4 Let’s Define the Loss Function.

- The Loss function used to learn the weights for Classification task is called Categorical Cross Entropy Loss and we will be using the same, which is defined as:

The formula for **cross entropy loss** is:

$$L_{CE}(y, \hat{y}) = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

Where:

- y is the **true label** from provided set of data.
- $\hat{y}$ is the **predicted label** by the classifier.
- C is the **number of classes** in dataset.

1.5 Formulating an Optimization Problem.

- **Multi layer Neural Network or DNN as Model Fitting Problem:**
  - **ERM Objective {Explicitly for DNN with Softmax in Output Layer}:**
    - The objective is to minimize the average loss (empirical risk) over the training dataset:
      - $\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{C} y_{ik} \log(\hat{y}_{ik})$;
    - Substituting $\hat{y}_{ik} = \frac{\exp(z_i)}{\sum_{k=1}^{C}\exp(z_{ik})}$, the loss can be written as:
      - $\mathcal{L}(\mathbf{W}, \mathbf{b}) = -\frac{1}{n}\sum_{i=1}^{n} \log\left(\frac{\exp(z_i)}{\sum_{k=1}^{C}\exp(z_{ik})}\right)$;
    - here $\mathbf{z_i} = \mathbf{W}^T\mathbf{x_i} + \mathbf{b}$; $\mathbf{W} \in \mathbb{R}^{d\times C} \rightarrow$ is the weight matrix and $\mathbf{b} \in \mathbb{R}^{C} \rightarrow$ is the bias vector.
  - **Formulating as an Optimization problem:**
    - For any parameter(s) $\rightarrow \theta^* = \left[\mathbf{w}, \mathbf{b}: \mathbf{w} \in \mathbb{R}^{d}, \mathbf{b} \in \mathbb{R}\right] \in \Theta$:
      - $\theta^* = \min_{\theta^*} \mathcal{L}(\mathbf{w}, \mathbf{b})$
    - This means **finding** the *weight vector* $\mathbf{w} \in \mathbb{R}^{d\times C}$ and *bias term* $\mathbf{b} \in \mathbb{R}^{C}$ that *minimize the average log loss* over the *training data*.

2. Computing Gradients.

{ Forward and Backward Propagation with Gradient Descent.}

2.1 Computing Gradients: Forward and Backward Propagations.

- The weights in **Multi layer networks** are learned with the **combinations** of **forward and backward propagations.**
  - a network forward propagates activation to produce an output and it backward propagates error to determine weight changes

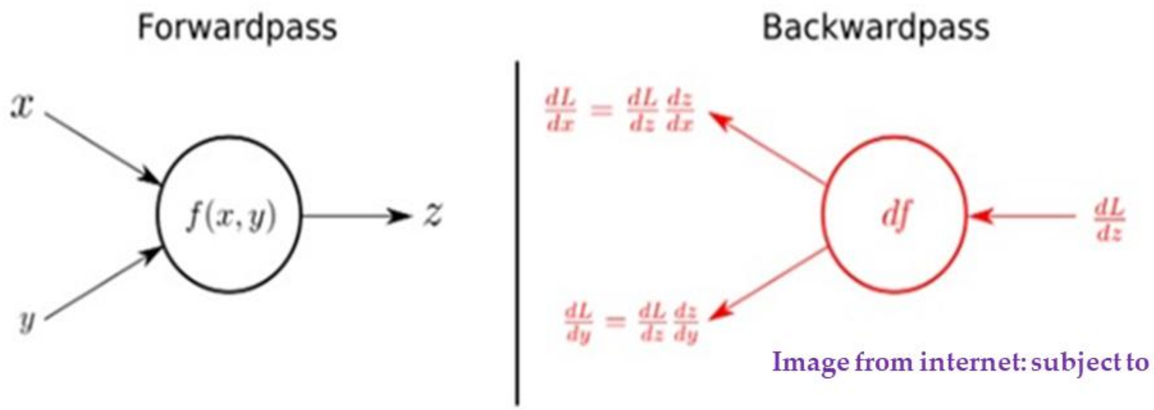
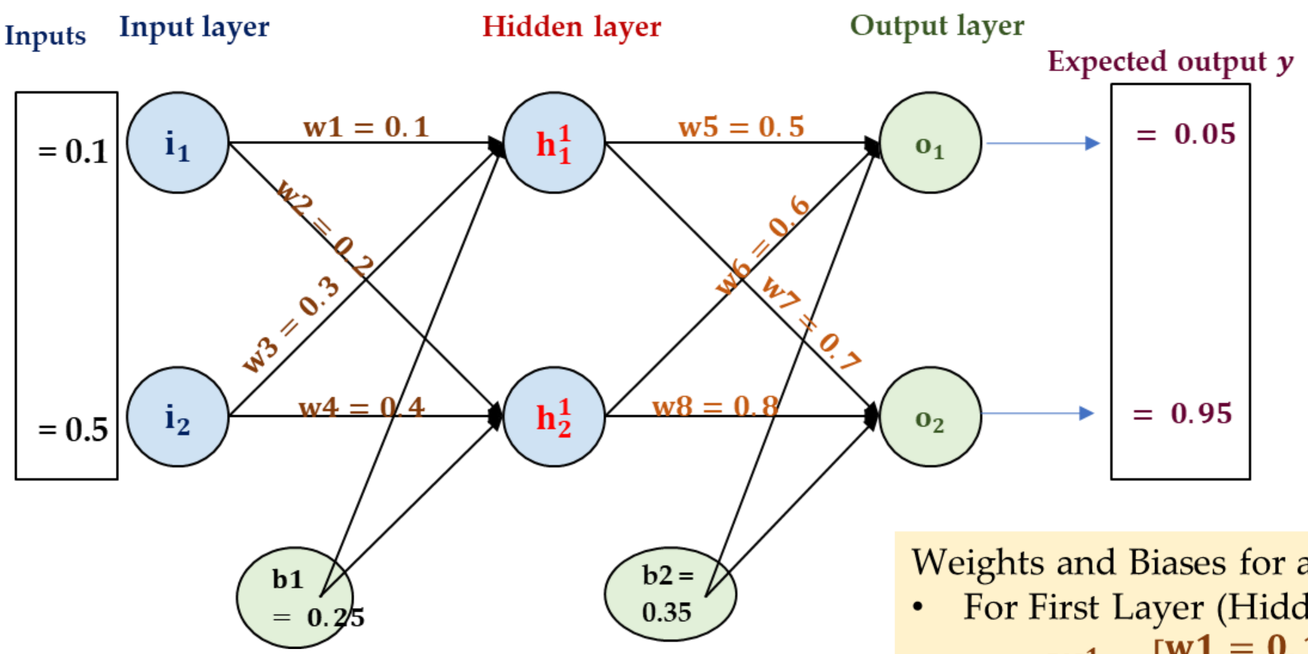


Image from internet: subject to copyright.

- Let's Understand by Solving with Hands on Example.

2.2 Solve the Neural Network Below:



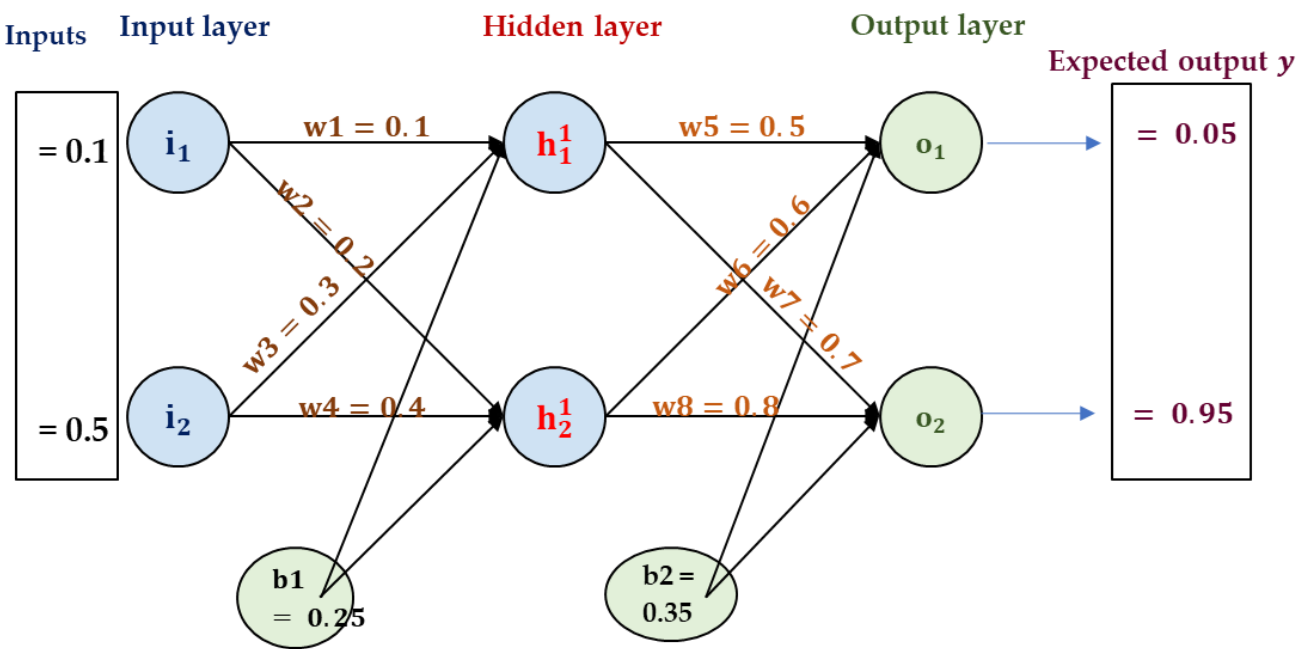
The Network Specification:

- The **Neural Network has three layers**:
  - Input Layer with two inputs.
  - **1 Hidden Layer** with **two sigmoid neurons**.
  - **1 Output Layer** with **two sigmoid neurons**.
    - {*Used Sigmoid for easy computation and better demonstration*}

Weights and Biases for all the Connections are denoted as:

- For First Layer (Hidden Layer):
  - $\mathbf{W^1} = \begin{bmatrix} \mathbf{w1 = 0.1} & \mathbf{w2 = 0.3} \\ \mathbf{w3 = 0.2} & \mathbf{w4 = 0.4} \end{bmatrix}$
- For Second Layer (Output Layer):
  - $\mathbf{W^2} = \begin{bmatrix} \mathbf{w5 = 0.5} & \mathbf{w6 = 0.7} \\ \mathbf{w7 = 0.6} & \mathbf{w8 = 0.8} \end{bmatrix}$
- Biases for hidden and output layers are:
  - $\mathbf{b1 = 0.25}$ $\rightarrow$ **Hidden Layers**
  - $\mathbf{b2 = 0.35}$ $\rightarrow$ **Output Layers**

2.2 Solve the Neural Network Below:



To – Do:

- Using two inputs $\mathbf{i_1}$ and $\mathbf{i_2}$:
  - Perform a forward pass through the network and **compute total error**.
  - Perform a backward pass to **propagate the error** within the network and **update the weights accordingly**.

2.3 Start the Solution:

Network Components

Input Layer:
$x_1 = 0.1,\ x_2 = 0.5$

Hidden Layer (2 neurons, Sigmoid activation):

Weights:

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.3 \\ 0.2 & 0.4 \end{bmatrix}$$

Bias: $b_1 = 0.25$

Output Layer (2 neurons, Sigmoid activation):

Weights:

$$W_2 = \begin{bmatrix} w_5 & w_6 \\ w_7 & w_8 \end{bmatrix} = \begin{bmatrix} 0.5 & 0.7 \\ 0.6 & 0.8 \end{bmatrix}$$

Bias: $b_2 = 0.35$

Expected Output:
$y_1^{\text{target}} = 0.05,\ y_2^{\text{target}} = 0.95$

2.4 Start the Computation from Hidden Layer:

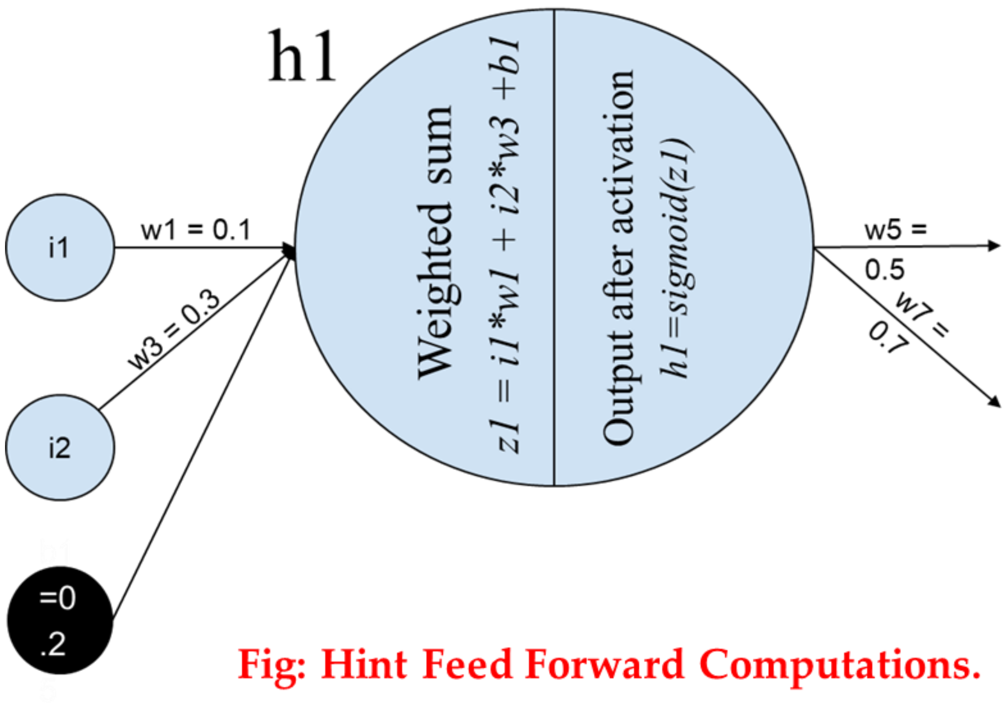


Fig: Hint Feed Forward Computations.

2.4.1 Sample Computations.

Each hidden neuron receives the weighted sum of inputs plus bias and applies the sigmoid function.

Compute Net Input to Hidden Neurons:

For hidden neuron h_1:

$$z_1 = (0.1 \times 0.1) + (0.5 \times 0.3) + 0.25$$
$$z_1 = 0.01 + 0.15 + 0.25 = 0.41$$

For hidden neuron h_2:

$$z_2 = (0.1 \times 0.2) + (0.5 \times 0.4) + 0.25$$
$$z_2 = 0.02 + 0.20 + 0.25 = 0.47$$

Apply Sigmoid Activation:

The Sigmoid function is:

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

For h_1:

$$h_1 = \frac{1}{1 + e^{-0.41}} \approx 0.601$$

For h_2:

$$h_2 = \frac{1}{1 + e^{-0.47}} \approx 0.615$$

2.4.2 Computations.

Step 3: Compute Output Layer Activations

Each output neuron receives the weighted sum of hidden layer outputs plus bias.

Compute Net Input to Output Neurons:

For output neuron y_1:

$$z_3 = ?$$

For output neuron y_2:

$$z_4 = ?$$

Apply Sigmoid Activation:

The Sigmoid function is:

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

For y_1:

$$y_1 = ?$$

For y_2:

$$y_2 = ?$$

2.4.2 Sample Solutions:

Each output neuron receives the weighted sum of hidden layer outputs plus bias.

Compute Net Input to Output Neurons:

For output neuron y_1:

$$z_3 = (0.601 \times 0.5) + (0.615 \times 0.7) + 0.35$$
$$z_3 = 0.3005 + 0.4305 + 0.35 = 1.081$$

For output neuron y_2:

$$z_4 = (0.601 \times 0.6) + (0.615 \times 0.8) + 0.35$$
$$z_4 = 0.3606 + 0.4920 + 0.35 = 1.2026$$

Apply Sigmoid Activation:

The Sigmoid function is:

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

For y_1:

$$y_1 = \frac{1}{1 + e^{-1.081}} \approx 0.7467$$

For y_2:

$$y_2 = \frac{1}{1 + e^{-1.2026}} \approx 0.7689$$

2.4.3 Compute Total Error.

Using the squared error function:

$$E = \frac{1}{2} \sum (y_{\text{target}} - y)^2$$

Total error:

$$\mathbf{E} = \mathbf{E_1} + \mathbf{E_2}$$

For y_1:

$$E_1 = ?$$

For y_2:

$$E_2 = ?$$

Total error:

$$E = E_1 + E_2 = ?$$

2.4.3 Sample Solution.

Using the squared error function:

$$E = \frac{1}{2}\sum(y_{\text{target}} - y)^2$$

Total error:

$$\mathbf{E} = \mathbf{E_1} + \mathbf{E_2}$$

For y_1:

$$E_1 = \frac{1}{2}(0.05 - 0.7467)^2$$

$$E_1 = \frac{1}{2}(-0.6967)^2$$

$$E_1 = \frac{1}{2}(0.4854) = 0.2427$$

For y_2:

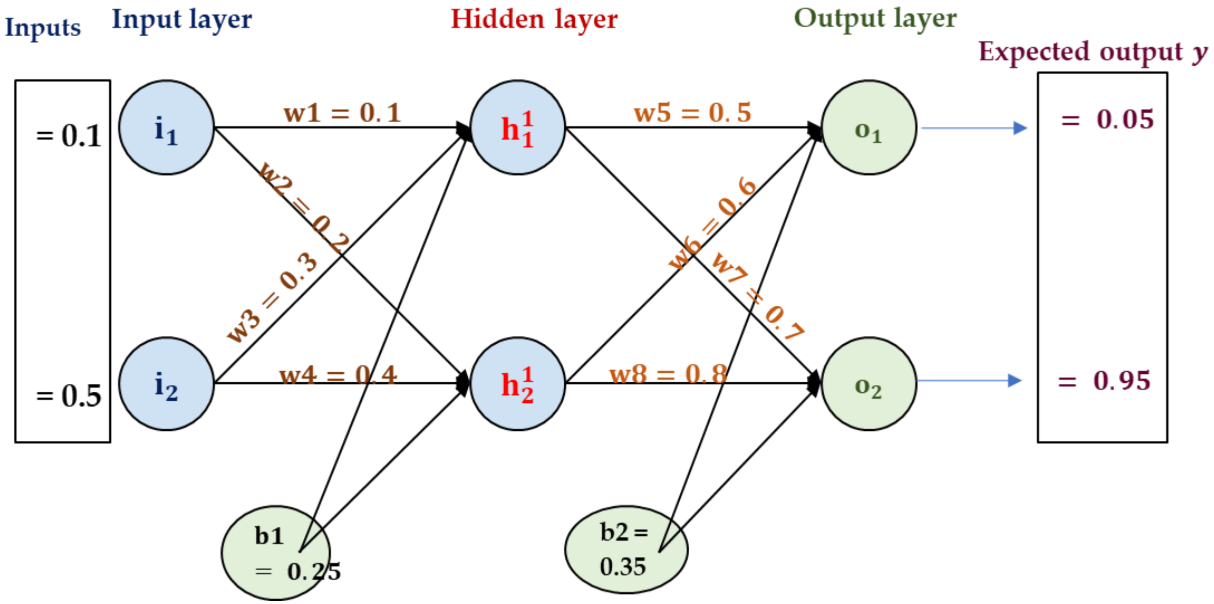
$$E_2 = \frac{1}{2}(0.95 - 0.7689)^2$$

$$E_2 = \frac{1}{2}(0.1811)^2$$

$$E_2 = \frac{1}{2}(0.0328) = 0.0164$$

Total error:

$$E = E_1 + E_2 = 0.2427 + 0.0164 = 0.2591$$

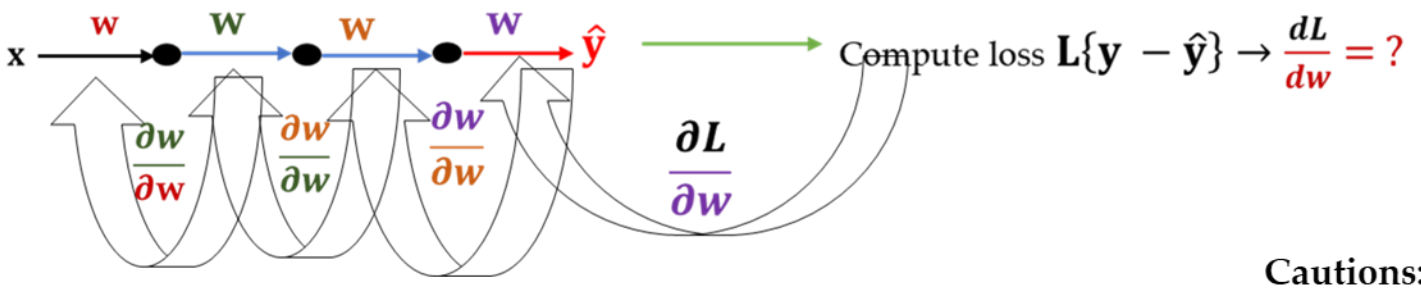


Congratulations you completed Forward Pass!!!!

3.1 Backward Propagation – Idea.

- Let's look into following network architecture:

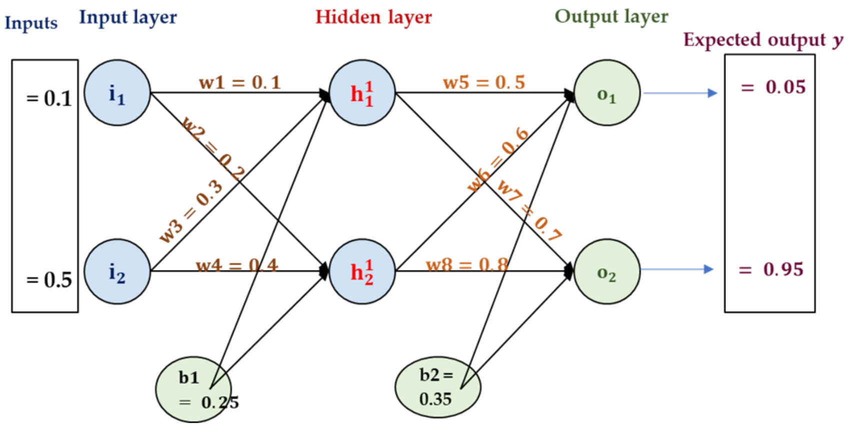
- Now we want to update our weight at first layer **w** using gradient descent for which we need to compute:
  - $\frac{dL}{dw} = ?$
- We can compute such using chain rule of derivative as:
  - $\frac{dL}{dw} = \frac{\partial L}{\partial w} \cdot \frac{\partial w}{\partial w} \cdot \frac{\partial w}{\partial w} \cdot \frac{\partial w}{\partial \mathbf{w}}$



Cautions: Only for demonstration purposes.

3.2 Backward Propagation – For our Network

- The purpose of the backward pass, also known as Backpropagation, is to distribute the overall error across the network .
- Modify the weights to minimize the cost function (loss). The weights are updated in a manner that ensures that the subsequent forward pass employs the updated weights.
- Decrease the overall error by a specific margin until the minima is achieved.
- Computes how much contribution each weight has on corresponding error.
- If we closely look at the example neural network, we can see that $E1$ is affected by $output_{o1}$, $output_{o1}$ is affected by sum_{o1} , and sum_{o1} is affected by w5.



3.3 Computing Partial Derivative of Error against W5

To compute the partial derivative of the error function E_{total} with respect to weight w_5, we use the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_5} = \frac{\partial E_{\text{total}}}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

This formula breaks down as follows:

Step 1: Compute $\frac{\partial E_{\text{total}}}{\partial y_1}$

The total error function is the sum of squared errors for all outputs. For simplicity, we focus on a single output error:

$$E_{\text{total}} = \frac{1}{2} \sum_i (y_i - y_i^{\text{target}})^2$$

For the first output neuron, the partial derivative is:

$$\frac{\partial E_{\text{total}}}{\partial y_1} = (y_1 - y_1^{\text{target}})$$

Step 2: Compute $\frac{\partial y_1}{\partial z_3}$

The output y_1 is the sigmoid function of the weighted sum z_3:

$$y_1 = \sigma(z_3) = \frac{1}{1 + e^{-z_3}}$$

The derivative of the sigmoid function is:

$$\frac{\partial y_1}{\partial z_3} = y_1(1 - y_1)$$

3.3 Computing Partial Derivative of Error against Weight

Step 3: Compute $\frac{\partial z_3}{\partial w_5}$
The weighted sum z_3 for the output neuron is:

$$z_3 = w_5 \cdot o_1 + w_6 \cdot o_2 + b_2$$

The partial derivative of z_3 with respect to w_5 is:

$$\frac{\partial z_3}{\partial w_5} = o_1$$

Step 4: Combine Everything
Now, we can combine all the pieces using the chain rule:

$$\frac{\partial E_{\text{total}}}{\partial w_5} = (y_1 - y_1^{\text{target}}) \cdot y_1(1 - y_1) \cdot o_1$$

This is the gradient of the error with respect to weight w_5.

3.3.1 The Computations.

Step 1: Compute $\frac{\partial E_{total}}{\partial y_1}$

The total error function E_{total} is the sum of squared errors for each output neuron. For the first output neuron, the partial derivative with respect to y_1 is:

$$\frac{\partial E_{total}}{\partial y_1} = (y_1 - y_1^{target})$$

Given that:

$$y_1 = 0.746, \quad y_1^{target} = 0.05$$

$$\frac{\partial E_{total}}{\partial y_1} = (0.746 - 0.05) = 0.696$$

3.3.1 The Computations.

Step 2: Compute $\frac{\partial y_1}{\partial z_3}$

The output y_1 is the sigmoid activation of the weighted sum z_3:

$$y_1 = \sigma(z_3) = \frac{1}{1 + e^{-z_3}}$$

The derivative of the sigmoid function is:

$$\frac{\partial y_1}{\partial z_3} = y_1(1 - y_1)$$

Given that:

$$y_1 = 0.746$$

$$\frac{\partial y_1}{\partial z_3} = 0.746 \times (1 - 0.746) = 0.746 \times 0.254 = 0.189$$

3.3.1 The Computations.

Step 3: Compute $\frac{\partial z_3}{\partial w_5}$

The weighted sum for the output neuron z_3 is:

$$z_3 = w_5 \cdot o_1 + w_6 \cdot o_2 + b_2$$

Thus, the partial derivative of z_3 with respect to w_5 is:

$$\frac{\partial z_3}{\partial w_5} = o_1$$

Given that:

$$o_1 = 0.593$$

$$\frac{\partial z_3}{\partial w_5} = 0.593$$

3.3.1 The Computations.

Step 4: Combine Everything to Compute the Gradient

Now, using the chain rule, we combine all the pieces:

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial y_1} \cdot \frac{\partial y_1}{\partial z_3} \cdot \frac{\partial z_3}{\partial w_5}$$

Substituting the values:

$$\frac{\partial E_{total}}{\partial w_5} = 0.696 \cdot 0.189 \cdot 0.593 \approx 0.078$$

3.3.1 The Computations.

Step 5: Weight Update for w_5

The weight update rule is given by:

$$w_5 = w_5 - \eta \cdot \frac{\partial E_{total}}{\partial w_5}$$

Assuming the learning rate $\eta = 0.1$, we compute the weight update:

$$w_5 = 0.5 \quad \text{(initial weight)}$$

$$w_5 = 0.5 - 0.1 \cdot 0.078$$

$$w_5 = 0.5 - 0.0078 = 0.4922$$

3.4 Computing Partial Derivative of Error against weights in second Layer.

- Repeat the Computations as earlier for w6, w7 and w8.
- Also update the weights.
    - { If you need take a help from Hint provided.}

3.5 Repeat the Computations for the weight in First Layer.

- Following is the example computations for w1:

Gradient Calculation for w_1

Gradient for w_1:
The weight update for w_1 can be calculated as:

$$\frac{\partial E_{\text{total}}}{\partial w_1} = \frac{\partial E_{\text{total}}}{\partial o_1} \cdot \frac{\partial o_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

Where:
- $o_1 = \sigma(z_1)$ is the output of the first hidden layer. - $z_1 = w_1 \cdot x_1 + w_2 \cdot x_2 + b_1$ is the weighted sum before applying the activation function.

3.5.1 Following is the example computations for w1:

Step 1: Calculate $\frac{\partial E_{\text{total}}}{\partial o_1}$

Step 1: Calculate $\frac{\partial E_{\text{total}}}{\partial o_1}$:
We start with the contribution of o_1 to the total error, which is:

$$\frac{\partial E_{\text{total}}}{\partial o_1} = \frac{\partial E_{\text{total}}}{\partial y_1} \cdot \frac{\partial y_1}{\partial o_1}$$

Given:
- $\frac{\partial E_{\text{total}}}{\partial y_1} = 0.696$ - The partial derivative $\frac{\partial y_1}{\partial o_1}$ is:

$$\frac{\partial y_1}{\partial o_1} = w_5 \cdot \sigma'(z_3) = w_5 \cdot y_1 \cdot (1 - y_1)$$

Substituting the values:
- $y_1 = 0.746$ - $w_5 = 0.4922$
Thus:

$$\frac{\partial y_1}{\partial o_1} = 0.4922 \cdot 0.746 \cdot (1 - 0.746) = 0.4922 \cdot 0.746 \cdot 0.254 \approx 0.092$$

Now, the total derivative with respect to o_1 is:

$$\frac{\partial E_{\text{total}}}{\partial o_1} = 0.696 \cdot 0.092 \approx 0.064$$

3.5.1 Following is the example computations for w1:

Step 2: Calculate $\frac{\partial o_1}{\partial z_1}$

Step 2: Calculate $\frac{\partial o_1}{\partial z_1}$:
The derivative of the sigmoid function is:

$$\frac{\partial o_1}{\partial z_1} = o_1 \cdot (1 - o_1)$$

Given:
- $o_1 = 0.593$
Thus:

$$\frac{\partial o_1}{\partial z_1} = 0.593 \cdot (1 - 0.593) = 0.593 \cdot 0.407 \approx 0.241$$

3.5.1 Following is the example computations for w1:

Step 3: Calculate $\frac{\partial z_1}{\partial w_1}$

Step 3: Calculate $\frac{\partial z_1}{\partial w_1}$:
The weighted sum for z_1 is:

$$z_1 = w_1 \cdot x_1 + w_2 \cdot x_2 + b_1$$

Thus:

$$\frac{\partial z_1}{\partial w_1} = x_1$$

Given:
- $x_1 = 0.5$
Thus:

$$\frac{\partial z_1}{\partial w_1} = 0.5$$

3.5.1 Following is the example computations for w1:

Combine Everything for w_1

Combine Everything for w_1:
Now, we can compute the gradient for w_1:

$$\frac{\partial E_{\text{total}}}{\partial w_1} = 0.064 \cdot 0.241 \cdot 0.5 \approx 0.0077$$

3.5.1 Following is the example computations for w1:

Weight Update for w_1

Weight Update for w_1:
Using the learning rate $\eta = 0.1$, the update for w_1 is:

$$w_1 = w_1 - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_1}$$

Given:
- $w_1 = 0.1$ (initial value)

Thus:

$$w_1 = 0.1 - 0.1 \cdot 0.0077 = 0.1 - 0.00077 = 0.09923$$

The updated weight w_1 is approximately $\boxed{0.09923}$.

3.6 Computing Partial Derivative of Error against weights in First Layer.

- Repeat the Computations as earlier for w2, w3 and w4.
- Also update the weights.
    - { If you need take a help from Hint provided.}

Congratulations you completed a Backpropagation!!!

- Before you come for your workshop, Please Think of an Idea on:
  - **Q: How can you build a program in python for above operations, so similar calculations can be repeated for number of epochs?**

Thank You