Neural Network

Perceptron

LAB-1 matlab using perception for And function with bipolar inputs and targets. Convergence curve & decision boundary show.

এটি একটি system যা classification করে বা feature detect করে।

~~এখানে~~ Input data গুলোর and training data এর উপর depend করে classify করবে যে input data গুলো কোন class এ আছে।

Neural Network:

Input, weight, Linear Combiner/summation, Activation function

Perceptron supervised learning এর under এ পড়ে।

perceptron এর parts.

Input, weight & bias, Summation, Activation function,

Perceptron Algorithm:

1. Initialize input-vector, weight, bias, desired-output, learning rate.

2. Predicted output of perceptron.

$$y(n) = sgn[weight * input + bias]$$

3. If $y(n)$ = desired-output then stop.

   Else ~~ud~~ update the weight:

$$y(n+1) = w(n) + \text{learning rate}[\text{desired-output} - \text{predicted output}] * \text{input vector}.$$

4. Go to step 2 until weight update stops.

For bipolar AND:

x_1	x_2	T
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

Let us assume. $\omega_1 = 0$, $\omega_2 = 0$

bias = 0

Learning rate $\eta = 1$

Threshold = 0

[illegible], bipolar, $\text{Sign}(\theta) \geq 0 = +1$, $< 0 = -1$

1st step:

$x_1 = -1$, $x_2 = -1$

~~$\Sigma w_i x_i = 0(-1) + 0(-1)$~~

~~$= 0$~~

Output, $Y_1 = \text{sign}(\omega_1 x_1 + \omega_2 x_2 + b)$ → sign function returns +1 if input is greater than or equal 0

$= \text{Sign}(0+0+0)$

$= \text{Sign}(0)$

$= +1$ here Target Output $T = -1$

এখানে [illegible] সাধারণত Threshold = 0
Threshold এর সমান বা বড় হলে +1
ছোট হলে -1

$\therefore Y_1 \neq T$ ∴ we need to update weight

∴ weight update formula

$\omega_i = \omega_i + \eta\,[\text{Target} - \text{Output}] * x_i$

bias update formula

$b = b + \eta * (T - d)$

$= 0 + 1(-1+1)$

$= -2$

$\therefore \omega_1 = 0 + 1[-1-1] * (-1)$

$= -2(-1) = 2$

$\omega_2 = 0 + 1[-1-1] * (-1)$

$= 2$

Again, the output of 1st step,

$Y_1 = \text{Sign}(\omega_1 x_1 + \omega_2 x_2 + b)$

$= \text{Sign}(2(-1) + 2(-1) - 2)$

$= \text{Sig}(-2-2-2) = \text{Sign}(-6) = -1$ ∴ $Y = \text{Target}$

∴ No weight update needs

Step-2:

$w_1 = w_2 = 2$, $b = -2$

Output, $y_2 = \text{sign}(w_1 x_1 + w_2 x_2 + b)$

$= \text{Sign}[2*(-1) + 2(+1) + 2]$

$= \text{Sign}[-2 + 2 + 2]$

$= \text{Sign}[2]$

$= +1$

∴ Output = Target hence no need to update weights and bias.

Step 3:

Output, $y_3 = \text{Sign}(w_1 x_1 + w_2 x_2 + b)$

$= \text{Sign}[2(1) + 2(-1) - 2]$

$= \text{Sign}(2 - 2 - 2)$

$= \text{Sign}(-2)$

$= -1$

∴ output = Target, so no update required

Step 4:

Output, $y_4 = \text{Sign}(w_1 x_1 + w_2 x_2 + b)$

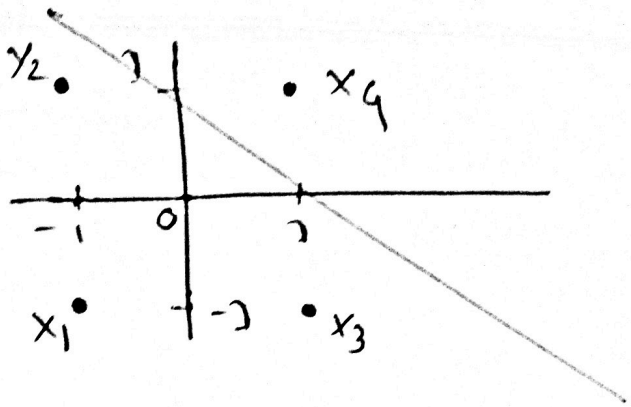
$= \text{Sign}(2(1) + 2(1) - 2)$

$= \text{Sign}[2 + 2 - 2]$

$= \text{Sign}(2)$

$= +1$ ∴ Output = Target

$x = -1, -1$
$-1, +1$
$+1, -1$
$+1, +1$



Convergence curve: It is a graphical representation that shows how the error decreases during the training process of ML model, such as perceptron. It plots the total error or loss (on y-axis) against nb of epochs (on the x-axis)

↓

Training iteration.

Decision boundary: It is a line that separates different classes in a classification problem. It typically show the input points and the decision boundary to see how the perceptron separates the different classes.

LAB 1

MATLAB using perceptron net for AND function.

Perceptron: In ML, perceptron is an algorithm used for supervised learning of binary classifiers. It determines whether or not an input belong to some specific class. It is a type of linear classifier.

Perceptron Algorithm:

1. Initialize input_vector, weight, bias, desired output & learning rate.
2. Predicted output, $Y(n) = \text{Sign}(\text{weight} * \text{input} + \text{bias})$
3. If desired output = Predicted output then stop.

   Else update weight

   $$Y(n+1) = w(n) + \text{learning rate}[\text{desired\_output} - \text{Predicted output}] * \text{input}$$
4. Go to step 2 until weight updates stop.

Convergence curve: It is a graphical representation that shows how the error decreases during the training process of ML model such as perceptron. It plots the total error/loss (on Y-axis) against nb of epochs (on x-axis)

Decision boundary: It is a line that separates different classes in a classification problem.

Bipolar AND

x_1	x_2	T
-1	-1	-1
-1	+1	-1
+1	-1	-1
+1	+1	+1

Assume, $\omega_1 = \omega_2 = b = 0$,

learning rate $\eta = 1$

~~Threshold = 0~~

Threshold θ

Activation function, step $= \begin{cases} -1, & \text{if } x < \theta \\ +1, & \text{if } x \geq 0 \end{cases}$

1st step:

$y_1 = \omega_1 x_1 + \omega_2 x_2 + b$

$= 0 + 0 + 0 = 0$

~~Act~~ $Y_1 = f_{step}(0) = +1$ but target is -1

$\therefore$ need to update weight

$\omega_i = \omega_i + \eta (T_0 - d_0) * x_i$

$\omega_1 = 0 + 1(-1-1) * (-1)$

$= -2(-1) = 2$

$\omega_2 = 0 + 1(-1-1) \times (-1)$

$= 2$

bias, $b = b + \eta (T_0 - d_0)$

$= 0 + 1(-1-2)$

$= -2$

$y_1 = 2(-1) + 2(-1) - 2$

$= -6$

$\therefore f_{step}(-6) = -1$ = Targeted output

hence no need to update weight

$y_2 = 2(-1) + 2(+1) - 2$

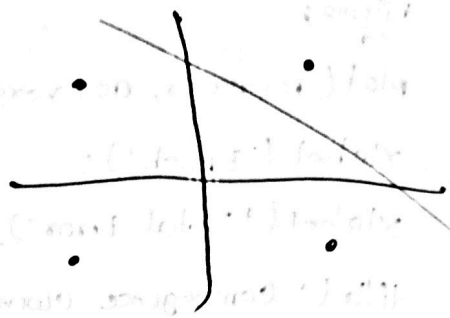
$= -2 + 2 - 2$

$= -2 = -1$ = Targeted output

$y_3 = 2(+1) + 2(-1) - 2$

$= 2 - 2 - 2$

$= -2 = -1$ = Targeted output



$y_4 = 2(+1) + 2(+1) + 2$

$= 2 + 2 + 2$

$= 6 = +1$ = Targeted output

Code

```
inputs = [-1,-1; -1,+1; +1,-1; +1, +1];

targets = [-1; -1; -1; +1];

weights = [0,0];

bias = 0;

eta = 1;

epochs = 10;

convergence_curve = zeros(epochs, 1);

for epoch = 1: epochs
    total_error = 0;
    for i=1: size(inputs, 1)
        x1 = inputs(i, 1);
        x2 = inputs(i,2);
        Output = sign(weights(1)*x1 + weights(2) *x2 + bias);
        if output == 0
            output = 1;
        end
        error = targets(i) - output;
        total_error = total_error + abs(error);
        if error ~= 0
            weights(1) = weights(1) + eta * error *x1;
            weights(2) = weights(2) + eta* error * x2;
            bias = bias+ eta *error;
end    end
```

```
plot(1:epochs, convergence_curve, '-o', 'LineWidth', 2);
xlabel('Epoch');
ylabel('Total Error');
title('Convergence Curve');
grid on;

figure;
hold on;
scatter(inputs(targets==-1,1), 'o', 'MarkerFaceColor', 'b',
```

sign() → 1 for positive nb
0 for zero
-1 for neg nb

2. Generate XOR function using McCulloch-pitts neuron

McCulloch-pitts model earliest ANN model.

Input could be either 0 or 1.

Theshold function as activation function.

McCulloch-pitts model can ~~take~~ have only two types of inputs.

1. Excitatory → positive magnitude
2. Inhibitory → negative

we can't directly get XOR using McCulloch-pitts neuron.

x_1 XOR x_2 = $x_1\bar{x}_2 + \bar{x}_1 x_2$

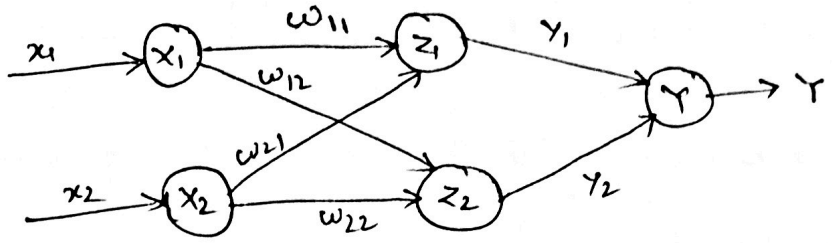
$= z_1 + z_2$

table-1

x_1	x_2	z_1
0	0	0
0	1	0
1	0	1
1	1	0

table-2

x_1	x_2	z_2
0	0	0
0	1	1
1	0	0
1	1	0



1st step:

Let consider weights $w_1 = w_2 = 1$

for table-1

for (0,0) input Y_1cal = 0

" (0,1) " " = +1

" (1,0) " " = +1

" (1,1) " " = +2

In this case we can't obtain threshold value so update weight. ~~in random manner~~

It can be in positive or neg magnitude.

Let $w_1 = +1$ and $w_2 = -1$

for (0,0) input, Z_1cal = 0

" (0,1) " , Z_1cal = -1

" (1,0) " , Z_1cal = 1

" (1,1) " , Z_1cal = 0

Here we can define a threshold value as +1

If $w_1 = +1$ and $w_2 = -1$ and

Threshold = +1 ,then we get the desired output.

Now, for table-2

using $w_1 = +1$, $w_2 = -1$ and Threshold = +1 we can't get the desired output.

now, let $w_1 = -1$, $w_2 = +1$

for (0,0) input Y_2cal = 0

for (0,1) input Y_2cal = +1

for (1,0) input Y_2cal = -1

for (1,1) input Y_2cal = 0

Here if we define Threshold = +1 and $w_1 = -1$ and $w_2 = +1$, then we get the desired output.

Now Final XOR table.

Y_1	Y_2	Y
0	0	0
0	1	1
1	0	1
0	0	0

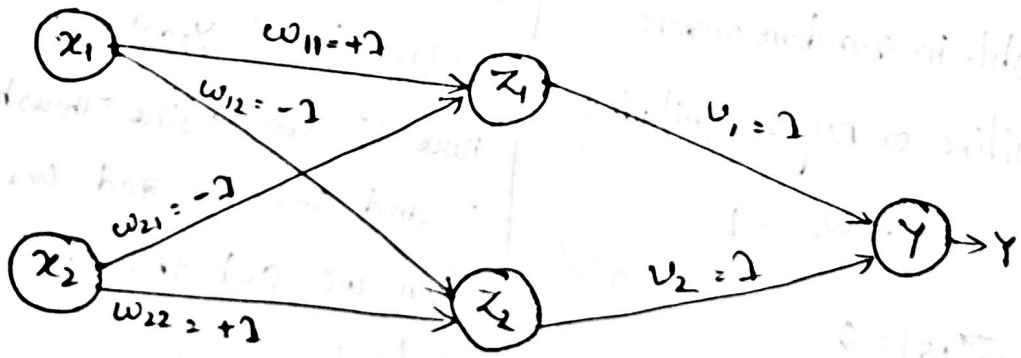
here let the weights $v_1 = v_2 = 1$

Then for, (0,0) . Ycal = 0

(0,1), Ycal = 1

(1,0), Ycal = 1

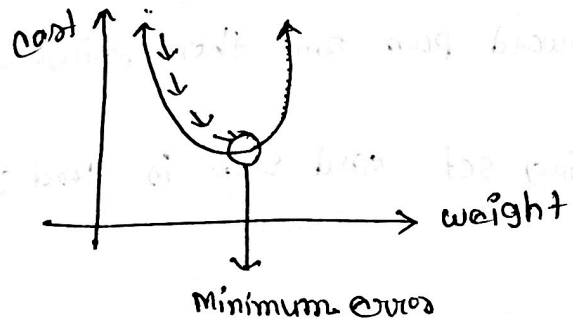
(0,0), Ycal = 0



3. Implement SGD method using Delta learning rule.

Delta rule: used for non-linearly separable data.

Delta rule use the gradient decend rule and find out the best weight



Each input example এর জন্য immidiately calculate করে.

Stochastic Guradient Descent এ sum করতে হয় না।
Guradiend Descent এ করতে হয়, before epoch এ 1 টা record নিয়ে output cal and then weight update. 1 point at a time → SGD

formula to modify weight:

$$w_i = w_i + \Delta w_i$$

$$\Delta w_i = -\eta \nabla E(w) \longrightarrow$$ Derivative of error w.r.t. weights/called also gradient

we can also write, ~~$\Delta w_i = \eta \Sigma (t_d - o_d) * x_i$~~

$$\Delta w_i = \eta (t_i - o_i) * x_i$$

Error function : $E_d(\vec{w}) = \frac{1}{2}(t_d - o_d)^2$

SGD machine learning model কে optimize করে,

4. Compare performance of SGD & Batch method.

- BGD use all training samples for one forward pass and then adjust weights.

  SGD use one sample for a forward pass and then adjust weights.

- BGD is good for small training set and SGD is good for big training set.

Gradient Descent weight and bias কে optimum করে যে accuracy maximum করে।

There are two important variants of Gradient ~~Decend~~ Descent that are widely used in Linear regression and NN.

They are,

1) Batch ~~Nor~~ Gradient Descent
2) Stochastic Gradient Descent

Batch Gradient Descent: It involves calculations over the full training set at each step. It is relatively slow on very large training data.

Stochastic Gradient Descent: In SGD, instead of using the entire dataset for each iteration, only a single random training example is selected to calculate the gradient and update the model parameters.

• Batch gradient descent takes longer to converge than stochastic gradient descent.

• Batch gradient descent is more accurate that SGD.

6. CNN for classification.

Image classification-এ ANN এর disadvantage হলো।

1. Too much computation.
2. Semitive to location of an object in an image.

CNN: Convolutional Neural Network is a type of deep learning neural network architecture used in computer vision. It is widely used in image classification, image recognition, face recognition, object recognition etc.

It has 1) Input Layer 2) Hidden Layer 3) Output layer.

Input Layer: It's the layer in which we give input to our model. The number of neuron in this layer is equal to the total number of features in our data.

Hidden Layer: The input from the input layer is then fed to the hidden layer. There could be many hidden layer based on our model and data size.

Output Layer: The output from the hidden layer is fed into a logistic function which classify the output in each class.

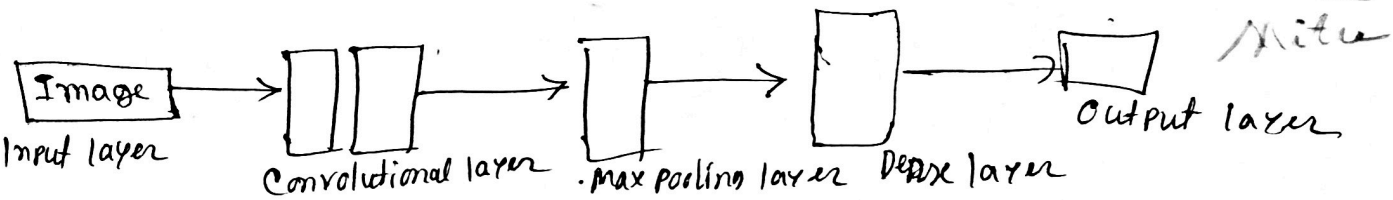


fig: CNN architecture

In Convolutional Layer ~~a input image~~ filter is applied into input image to detect specific features like edges, patterns.

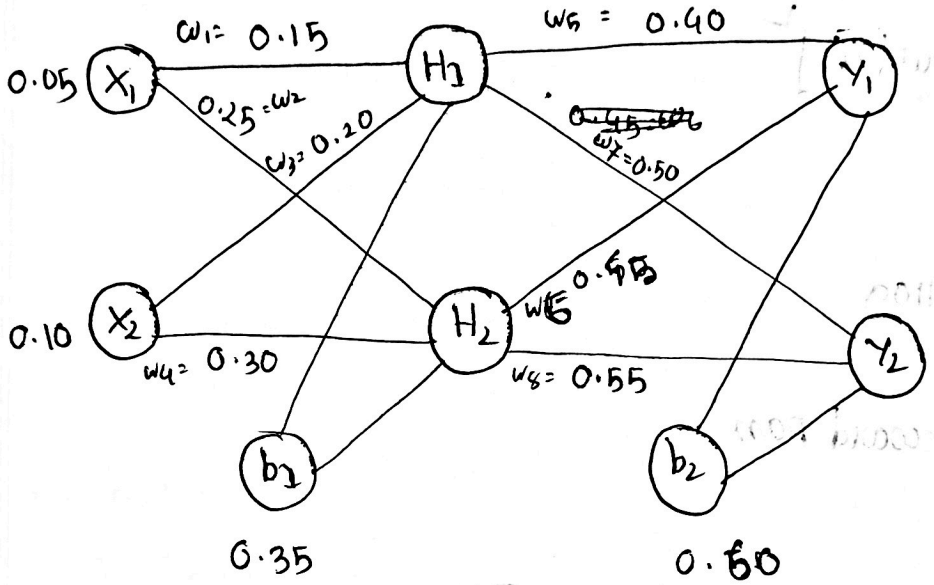
Maxpooling reduces the size of data and make network efficient

Dense layer combines the features learned by the ~~previous~~ previous layers to make the final prediction. Here each neuron is connected to every neuron in the previous layer.

7. BACKPROPAGATION

When error occurs, we go in back direction that is

Output → hidden → input layer



Activation function sigmoid.

Forward Pass:

$H_1 = x_1 w_1 + x_2 w_2 + b_1 = 0.3775$

$Out(H_1) = \frac{1}{1+e^{-H_1}} = 0.593269992$

$H_2 = x_1 w_3 + x_2 w_4 + b_1 = 0.3925$

$Out(H_2) = \frac{1}{1+e^{-H_2}} = 0.596884378$

$y_1 = Out(H_1) * w_5 + Out(H_2) * w_6 + b_2 = 1.105905967$

$Out(y_1) = \frac{1}{1+e^{-y_1}} = 0.75136507$

$y_2 = Out(H_1) \times w_7 + Out(H_2)\, w_8 + b_2 = 1.224921404$

$Out(y_2) = \frac{1}{1+e^{-y_2}} = 0.772928465$

Error Calculation:

$$E_1 = \frac{1}{2}\left[T_1 - Out(y_1)\right]^2$$

$$E_2 = \frac{1}{2}\left[T_1 - Out(y_2)\right]^2$$

$$E = E_1 + E_2$$

$$= 0.298371109$$

3rd Step: Backward pass

Consider, w_5

$$\frac{\partial E}{\partial w_5} = \frac{\partial}{\partial Out(y_1)} \times \frac{\partial Out(y_1)}{\partial y_1} \times \frac{\partial y_1}{\partial w_5}$$

Now, $$\frac{\partial E}{\partial Out(y_1)} = \frac{\partial}{\partial Out(y_1)}\left[\frac{1}{2}(T_1 - Out y_1)^2 + \frac{1}{2}(T_2 - Out y_2)^2\right]$$

$$= \frac{1}{2} \cdot 2\left[T_1 - Out(y_1)\right](0 - 1) + 0$$

$$= -T_1 + Out(y_1) = 0.74136507$$

~~$$\frac{\partial Out(y_1)}{\partial y_1} = \frac{\partial}{\partial y_1}\left(\frac{1}{1 + e^{-y_1}}\right)$$~~

$$\frac{\partial Out(y_1)}{\partial y_1} = Out(y_1) * \left[1 - Out(y_1)\right] = 0.186815602$$

$$\frac{\partial y_1}{\partial w_5} = Out(H_1)$$

$$= ~~0.186815602~~ \; 0.593269992$$

$\therefore \frac{\partial E}{\partial w_5} = 0.082167041$

$\eta = 0.5$ ধরে নেই

$w_5 = w_5 - \eta * \frac{\partial E}{\partial w_5}$

$= 0.40 - 0.5 * 0.082167041$

$= 0.35891648$

In same way, সব weight update করতে হবে।

Backpropagation is an iterative algorithm, that helps to reduce the cost function by determining which weights and biases should be adjusted.

During every epoch, the model learns by adapting the weights and biases to minimize the loss.

Backpropagation algorithm works by two different passes. They are-

1. Forward pass
2. Backward pass ⟶ Error is transmitted back to the network.

SVM

Support vector Machine is a powerful ML algorithm used for linear & non-linear classification, regression.

SVM can be used for -

1. Text classification.
2. Image "
3. Spam detection
4. Handwritten identification
5. Face detection.

Supervised ↳ machine is trained on a set of labeled data.

SVM is a supervised Machine learning that is best suited for classification.

The best boundary in SVM is known as hyperplane of SVM.

There are two types of SVM

1. Linear : used for linearly separable data
2. Non-Linear : used for non-linearly

PCA

~~Principal~~

Principal Component Analysis.

PCA is an unsupervised learning algorithm while the data into higher dimensional space is mapped to lower dimentional space. The main goal of PCA is to reduce the dimentionality of a dataset, as over dimentional data cause overfitting, increase computation time and reduce accuracy.

PCA steps

1st step: Calculate mean of each attribute.

Say there are two attribute x and y

So calculate $\bar{x}$ & $\bar{y}$

2nd step: Covariance matrix find

$$Cov(x,y) = \sum_{i=1}^{n} \frac{(x_i - \bar{x})(y_i - \bar{y})}{n-1}$$

3rd step: Compute Eigenvalue & Eigenvector.

$$C - \lambda I = 0$$

here C = Covariance matrix

λ = Eigenvalue

I = Identity matrix.

Again, $Cv = \lambda v$

here v = Eigenvector = $\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$