Single Layer Perceptron (SLP)

05 October 2025 09:55

Single Layer Perceptron

what is a perceptron?

Perception : derived from the word 'perceive'

- w.r.t. psychology / human context

Perception is the process by which we interpret and make sense of sensory information.

what we see, hear, touch, smell etc.

(machines + softwares)

In AI (Deep learning)

Perception refers to how machines interpret raw data from the world — turning it into meaningful information

PERCEP + TRON

PERCEP: derived from the word perception

TRON: some sort of electronics devices → GPUs.

— to mimicking the human intelligence using electronics.

① algorithms & frameworks (TensorFlow)

② Hardware such as computers with GPUs

(TensorFlow) + (electronics devices) → Perceptron based model.

SINGLE LAYER PERCEPTRON (SLP)

In machine learning, the perceptron is an algorithm for supervised learning of binary classifiers

History [edit]

See also: History of artificial intelligence § Perceptrons

[ANN was invented in 1943.]

The artificial neuron network was invented in 1943 by Warren McCulloch and Walter Pitts in *A logical calculus of the ideas immanent in nervous activity*.[5]

In 1957, Frank Rosenblatt was at the Cornell Aeronautical Laboratory. He simulated the perceptron on an IBM 704.[6][7] Later, he obtained funding by the Information Systems Branch of the United States Office of Naval Research and the Rome Air Development Center, to build a custom-made computer, the Mark I Perceptron. It was first publicly demonstrated on 23 June 1960.[8] The machine was "part of a previously secret four-year NPIC [the US' National Photographic Interpretation Center] effort from 1963 through 1966 to develop this algorithm into a useful tool for photo-interpreters".[9]

Rosenblatt described the details of the perceptron in a 1958 paper.[10] His organization of a perceptron is constructed of three kinds of cells ("units"): AI, AII, R, which stand for "projection", "association" and "response". He presented at the first international symposium on AI, *Mechanisation of Thought Processes*, which took place in 1958 November.[11]

https://en.wikipedia.org/wiki/Perceptron

SLP was developed by Frank Rosenblatt in 1958 to do a simple binary classification

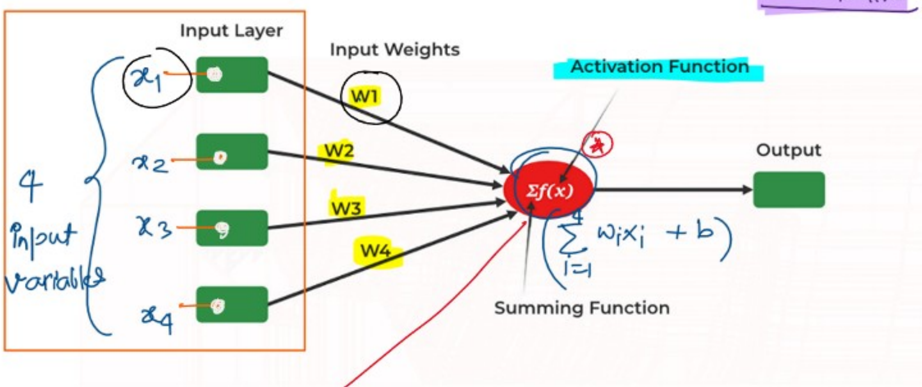
A SLP consists of only one layer of weights that directly connects the input features to the output

It is a feed-forward ANN with NO HIDDEN LAYER

* It is one of the simplest types of ANNs designed to mimic the way neurons work in the brain

Structure of a Single-Layer Perceptron (SLP)

Illustrative



Weighted sum of inputs

$$x/z = w_1 x_1 + w_2 x_2 + w_3 x_3 + w_4 x_4 + b$$

weights

bias

$$\boxed{x/z = \sum_{i=1}^{4} w_i x_i + b}$$

x/z → Activation Function (AF) $f(z)$ → Threshold t

(Separate section will be discussed later)

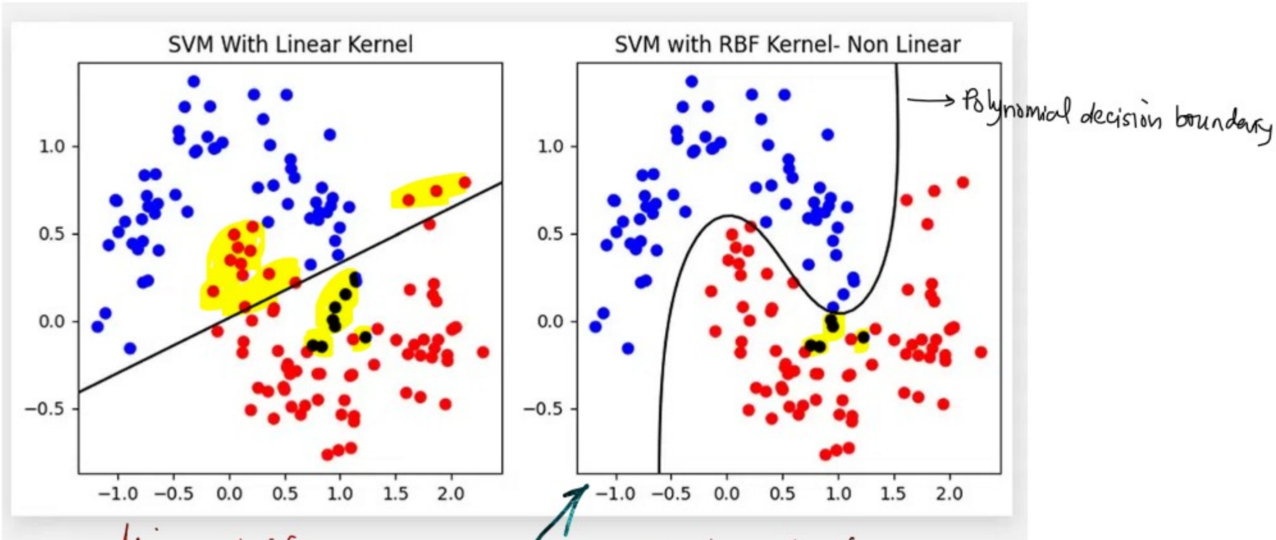
Output layer

$\geq t \rightarrow$ class #1

$< t \rightarrow$ class #0

In logistic regression $\begin{cases} \geq 0.5 \rightarrow \text{class \#1} \\ < 0.5 \rightarrow \text{class \#0} \end{cases}$

Activation Function (AF)



linear classifier   non-linear classifier

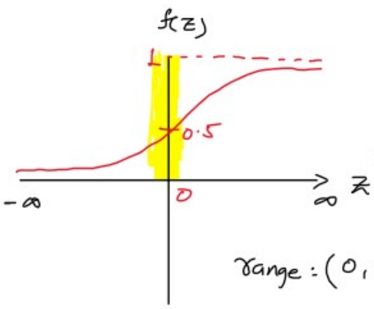
An activation function introduces non-linearity into a neural network model.

Without activation function, the neural network would just be a linear regression model, no matter how many layers are added.

lets the network learn complex, non-linear patterns like images, speech and text.

Sigmoid Function

$$f(z) = \left(\frac{1}{1+e^{-z}}\right)$$

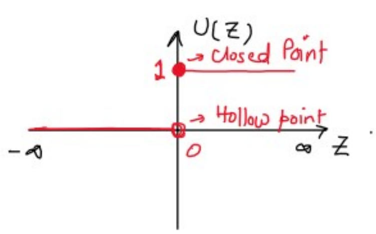


range : (0, 1)

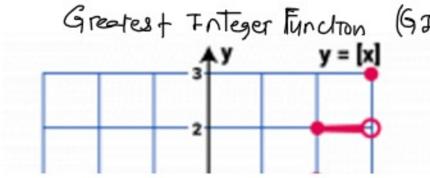
S-shaped curve

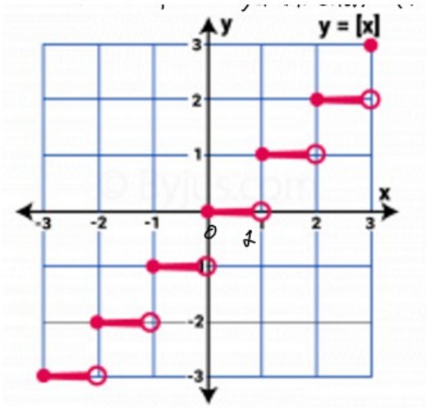
Unit step Activation Function

$$U(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$$



Greatest Integer Function (GIF)





$[x]$

$[0.3] = 0$

Building SLP using a tiny dataset (Binary classification)

SLP Demonstration using Excel

Let us take a tiny dataset having:

- two features say x_1 and x_2 - two input variables
- target or label → y values
- 4 training rows or training examples.

Feature 1 : X1	Feature 2 : X2	Label (Y)
2	1	1
1	-1	0
-1	-2	0
-2	1	1

→ We'll build SLP using Excel

→ and then move the SLP to Python using our own functions

PERCEPTRON LEARNING RULE

$$w_j := w_j - \alpha * (\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{Error}}) * x_j^{(i)}$$

$$b := b - \alpha * (\underbrace{\hat{y}^{(i)} - y^{(i)}}_{\text{Error}})$$

Given error = 0, both weight and bias need not be updated

$$w_1)_{new} = w_1)_{old} - \alpha * error * x_1$$

$$= w_1)_{old} - 0.1 * error * x_1$$

SLP demonstration using Excel

SINGLE LAYER PERCEPTRON (for only 1 Epoch)
Using Stochastic Gradient Descent (SGD) Optimizer

SINGLE LAYER PERCEPTRON (for only 1 Epoch)			
Using Stochastic Gradient Descent (SGD) Optimizer			

	Feature 1 : X1	Feature 2 : X2	Label (Y)
Row #1	2	1	1
Row #2	1	-1	0
Row #3	-1	-2	0
Row #4	-2	1	1

← (Actual / Target values)

Initial weights & bias

 # weights variables are equal to number of features

W1	W2	b
0.5	-0.5	0.1

← user-defined random weights and bias

Activation Function		
Unit Step Funtion	if z>=0 then 1 and if z<0 then 0	*Class 1 or Class 0*

Weighted sum of inputs + bias

ROW #1	Feature 1 : X1	Feature 2 : X2	Label (Y)
	2	1	1
WEIGHTED SUM OF INPUTS + BIAS	0.6		

← $z = (W_1 x_1 + W_2 x_2 + b)$

$z = 0.5 * 2 - 0.5 * 1 + 0.1$

$= 1 - 0.5 + 0.1 = 0.6$

PREDICTED Y_hat Class	*1*
ACTUAL CLASS	1
ERROR (Predicted - Actual)	0

convert the excel into a python based SLP model.

SLP Code Explanation

1. Load the data

```
]: X = np.array(
       [[2, 1],
        [1, -1],
        [-1, -2],
        [-2, 1]])

   print("Input Feature Array:", X)

   Input Feature Array: [[ 2  1]
    [ 1 -1]
    [-1 -2]
    [-2  1]]
```

```
]: Y = np.array([1, 0 , 0, 1])
   Y

]: array([1, 0, 0, 1])
```

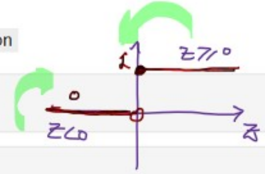
→ creating | hardcoding couple of arrays

or can use a separate function to generate data like we had done in BGD.

$\underbrace{\text{generate data}}_{\text{using Numpy}}$

2. Define the `activation function` ¶

`Unit Step Activation Function`



[6]:
```python
def step_af(z):
    return 1 if z>=0 else 0
```

[7]:
```python
step_af(0.000001)
```

[7]: 1

[8]:
```python
step_af(0)
```

[8]: 1

[9]:
```python
step_af(-0.0000000000000001)
```

[9]: 0

this function will measure how well the SLP is performing

how many classification mistakes (errors) it makes

```python
### 3.a. Compute Cost
def compute_cost(X,Y,W,b):
    '''
    X: Input Array
    Y: Target (Y labels)
    W: Weight Array
    b: bias
    '''
    total_errors = 0 # how many misclassifications have happened
```

initializing total_errors variable and setting it to zero

increments the variable every time a prediction ≠ actual.

Looping over all training examples | sample | rows.

```python
    for i in range(len(X)):
        z = np.dot(W, X[i]) + b # weighted sum of inputs + bias --> one row at a time
        y_pred = step_af(z) # predicted value using unit step activation function
        if y_pred != Y[i]: # comparing predicted Y against actual Y
            total_errors += 1 #counting the instances of error
    return total_errors
```

iterating over each training sample.

weighted sum of inputs + bias

$W_1 X_1^{(i)} + W_2 X_2^{(i)} + b \quad i = 0$

perceptron's forward pass

not equal

applying activation function to convert continuous z into a binary class → 0 → 1.

compare prediction with actual label.

total_errors = total_errors + 1 → if misclassification happens, total_errors increments by 1.

$$total\_errors = \underbrace{total\_errors}_{0} + 1$$

$= 0 + 1 = 1$

→ $1 + 1 = 2$

→ $2 + 1 = 3$.

after checking for all the samples, the function returns how many mistakes were made.

```python
#### 3.b. Let us create the SLP train

def slp_train(X, Y, W, b, alpha, epochs):
    ### Create the couple of empty lists to track historical updates
    errors_per_epoch = []
    accuracy_per_epoch = []

    ### Suppress scientific notation and control decimals
    np.set_printoptions(precision = 2, suppress = True)

    for epoch in range(epochs): # outer for loop
```

→ how many misclassifications occurred each epoch.

→ fraction of correctly classified samples each epoch.

```
Epoch# 1
Input:      [2 1]  Prediction:  1  Actual:  1  Error:  0  We
```

of correctly classified samples each epoch.

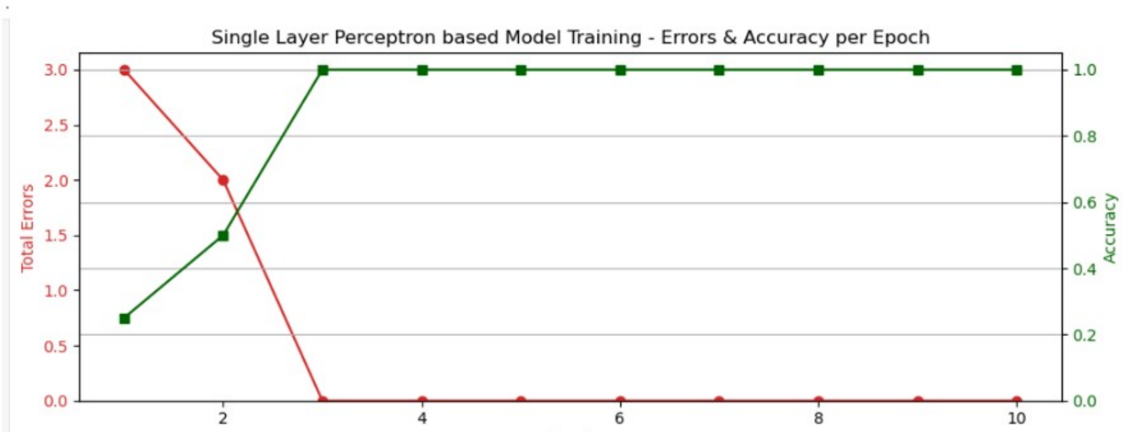
```python
### Suppress scientific notation and control decimals
np.set_printoptions(precision = 2, suppress = True)

for epoch in range(epochs): # outer for loop
    print(f"\nEpoch# {epoch + 1}")

    total_errors = 0 # instances of mis-classification
    correct = 0 #instances of correct classification
```

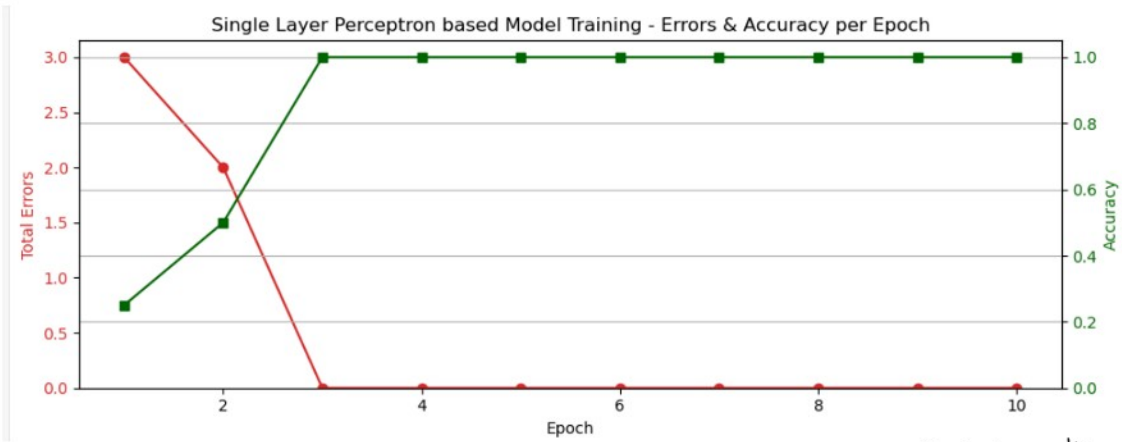
```
Epoch# 1
Input:     [2 1]  Prediction:  1  Actual:  1  Error:  0  We
Input:     [ 1 -1]  Prediction:  1  Actual:  0  Error:  1
Input:     [-1 -2]  Prediction:  1  Actual:  0  Error:  1
Input:     [-2  1]  Prediction:  0  Actual:  1  Error:  -1
```

accuracy-per_epoch = $\frac{1}{4}$ = 0.25.

```
Epoch# 1
Input:     [2 1]  Prediction:  1  Actual:  1  Error:  0  Weights:  [ 0.5 -0.5]  Bias:  0.100  Cost:  3
Input:     [ 1 -1]  Prediction:  1  Actual:  0  Error:  1  Weights:  [ 0.4 -0.4]  Bias:  0.000  Cost:  3
Input:     [-1 -2]  Prediction:  1  Actual:  0  Error:  1  Weights:  [ 0.5 -0.2]  Bias:  -0.100  Cost:  2
Input:     [-2  1]  Prediction:  0  Actual:  1  Error:  -1  Weights:  [ 0.3 -0.1]  Bias:  0.000  Cost:  2
```

```python
for epoch in range(epochs): # outer for loop
    print(f"\nEpoch# {epoch + 1}")

    total_errors = 0 # instances of mis-classification
    correct = 0 #instances of correct classification
    # } resetting the counters for every epoch.

    for i in range(len(X)): # inner for loop -> iterating through each row/Training sample.
        x = X[i] # picking a row - one at a time
        y = Y[i] # picking the target label for that specific row -- actual Y
        z = np.dot(W,x) + b # weighted sum of inputs + bias
        y_pred = step_af(z) # predicted Y
        error = y_pred - y # calculate error

        if error != 0:
            ### Since there is an error, need to update weights & biases
            W = W - (alpha * (error * x))
            b = b - (alpha * error)
            total_errors += 1 <- increment the counter when there is an error
        else:
            correct += 1 <- if there is no error, correct counter increments
        cost = compute_cost(X, Y, W, b) -> cost metric for the epoch.
        # Well-formatted log output
        print(f"{'Input:':<10} {x}  "
              f"{'Prediction:':<12} {y_pred}  "
              f"{'Actual:':<8} {y}  "
              f"{'Error:':<7} {error}  "
              f"{'Weights:':<9} {W}  "
              f"{'Bias:':<6} {b:.3f}  "
              f"{'Cost:':<6} {cost}")

    errors_per_epoch.append(total_errors) #appending the historical errors
    accuracy_per_epoch.append(correct/len(X)) #appending the avg. correct instances

return W, b, errors_per_epoch, accuracy_per_epoch
```





In principle, it's good to run more number of epochs

epochs ≠ 100 ??

Let us do plotting - `TASK - go through it first and then I will take it tmrw ???`

15]:

```python
###  === Plot errors and accuracy in a single chart using primary & secondary axes === ###
fig, ax1 = plt.subplots(figsize = (10,4))

color = 'tab:red'
ax1.set_xlabel('Epoch')
ax1.set_ylabel('Total Errors', color = color)
ax1.plot(range(1, epochs + 1), errors, marker='o', color=color, label='Errors')
ax1.tick_params(axis='y', labelcolor= color)
ax1.set_ylim(bottom = 0)

ax2 = ax1.twinx()
color = 'darkgreen'
ax2.set_ylabel('Accuracy', color = color)
ax2.plot(range(1, epochs + 1), accuracy, marker='s', color=color, label='Accuracy')
ax2.tick_params(axis='y', labelcolor= color)
ax2.set_ylim(0, 1.05)

plt.title("Single Layer Perceptron based Model Training - Errors & Accuracy per Epoch")
fig.tight_layout()
plt.grid(True)
plt.show()
```