Assignment Deep Learning

Name- Reetesh kumar Srivastava (M.Tech Data Science)

Enrollment number - 21/10/MI/008

Feed Forward Network

inputs=[1,X1,X2,.....Xn]

weights=[Wo,W1,W2....Wn]

Z=X0W0+X1W1+X2W2......XnWn

Z is summation of product of Input and their Associated weights

Step Function is used to decide output based on value of Z

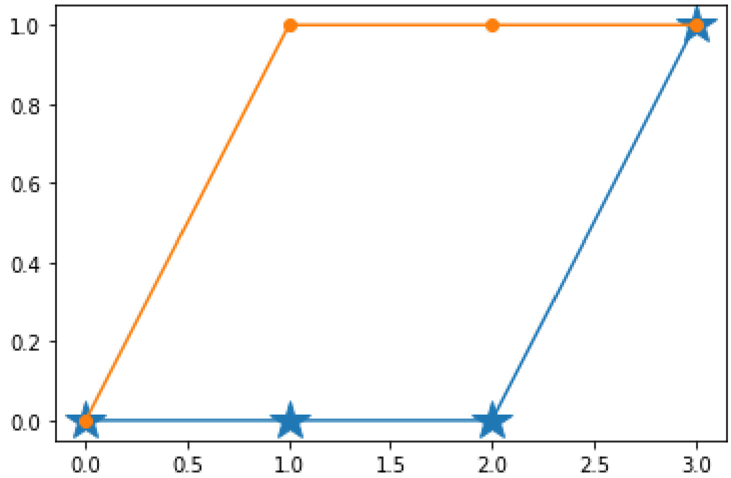
In [10]:

```
import numpy as np
import matplotlib.pyplot as plt
i_input=np.array([[0,0],[0,1],[1,0],[1,1]]) #input values of AND gate
labels=np.array([0,0,0,1])  #initially labeles for each input of i_input set
weights=[0.784,0.897]  #associated weights
threshold=0.54          #threshold value
# Defining step function
def step_fun(sum):
    if sum>threshold:
        return 1
    else:
        return 0
#iterating through i_input array to  calculate Z
updated_labels=[]
for i in range (0, i_input.shape[0]):
    actual_value=labels[i]
    instances=i_input[i]
    x0=instances[0]
    x1=instances[1]
    z=x0*weights[0]+x1*weights[1] # Z is sum of Product of Inputs and their ass
    fire= step_fun(z)
    updated_labels.append(fire)
    delta=actual_value-fire  #delta is Error (When Error is  0 it means predict
    print("Predicted value ", fire ," Whereas Actual Value", labels[i] ," Error
```

```
Predicted value  0  Whereas Actual Value 0  Error is  0
Predicted value  1  Whereas Actual Value 0  Error is  -1
Predicted value  1  Whereas Actual Value 0  Error is  -1
Predicted value  1  Whereas Actual Value 1  Error is  0
```

In [2]:
```
plt.plot(labels, marker='*', ms=20)
plt.plot(updated_labels, marker='o')
```

Out[2]:
```
[<matplotlib.lines.Line2D at 0x7f72c46b7e48>]
```



SUMMARY

We have a set of input along with actual output. Now this model associates some weights randomly in order to predict the output. We have can track whether our outcome is Correctly predicted or not with the help of Error (delta).

In above graph Orange Circles indicate Actual Value and Blue Stars indiacte Predicted Value . We can see that for 3rd input [1,0] Predicted and Actual outcomes vary

This Variation can be solved using Gradient Descent aprroach by using Learning rate for Weight Updation

Perceptron Training Rule

Learning Problem is to determine Weights that causes perceptron to produce correct output

delta- delta is the difference between Predicted and Actual outputs ¶

We keep on modifying weights whenever it misclassifies an example.Weights are modified at each step iteratively according to perceptron learning rate untill it classifies all training examples correctly

Wi=Wi+ ΔW

ΔW=η(t-o)Xi

η is positive learning rate. Role of η to moderate degree at which weights are changing

In [5]:
```python
import numpy as np
import matplotlib.pyplot as plt
i_input=np.array([[0,0],[0,1],[1,0],[1,1]]) #input values of AND gate
y=np.array([0,0,0,1])  #y is target output for each input of i_input set
w=[0.78,0.91]  #associated weights
threshold=0.54  #threshold value
iteration=5
eta=0.1       #eta is learning rate

# Defining step function
def step_fun(sum):
    if sum>threshold:
        return 1
    else:
        return 0
print("Initial Weights ", w)

#iterating through i_input array to  calculate Z
updated_labels=[]
for j in range(0,iteration):
    print("Iteration ",j)
    print("Actual(y)"," ","Predicted(y')"," ","Error")
    for i in range (0, i_input.shape[0]):
        actual_value=y[i]
        instances=i_input[i]
        x0=instances[0]
        x1=instances[1]
        z=x0*w[0]+x1*w[1] # Z is sum of Product of Inputs and their associated
        fire= step_fun(z)
        updated_labels.append(fire)
        delta=actual_value-fire  #delta is Error (When Error is  0 it means pre
        print( y[i],"  "*12,fire,"  "*12,delta)
        w[0]=w[0]+delta*eta #Updating Weights
        w[1]=w[1]+delta*eta
    print("_"*35)
print("Updated Weights after Iteration",w) #Updated Weights after learning
```

```
Initial Weights  [0.78, 0.91]
Iteration  0
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              1              -1
1              1              0
___________________________________
Iteration  1
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              0              0
1              1              0
___________________________________
Iteration  2
Actual(y)   Predicted(y')   Error
0              0              0
0              1              -1
0              0              0
1              1              0
___________________________________
Iteration  3
Actual(y)   Predicted(y')   Error
0              0              0
0              0              0
0              0              0
1              1              0
```

```
Iteration  4
Actual(y)   Predicted(y')   Error
0             0             0
0             0             0
0             0             0
1             1             0

Updated Weights after Iteration [0.3800000000000001, 0.5100000000000001]
```

Summary

Initially a random weight was chosen and the Two predicted outputs were misclassified.

After applying Perceptron Training Rule , Weights were Modified till it classified Examples correctly till some iteration

Initially weights was [0.78,0.91] after Updation [0.38, 0.51] and this updated weights predicted output Correctly after few iterations

Gradient Descent

Activation fun 1(/1+e^-weighted_sum)

weighted_sum=W1X1 + W2X2 +....Wi*Xi+Bias

Loss = -(targetlog(pred)+(1-target)log(1-pred))

Wi=Wi+ ΔW

ΔW=η(t-o)Xi

New Bias(b')= Old Bias(b) + η*(target-predicted)

η is Learning rate which ensures gradual weight update

Bias helps to tune our model .

In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt
def Activation_fun(z): #z is weighted sum of input and associated weights
    return 1/(1+np.e**-z)
def get_prediction(Input,Weights,bias):
    return Activation_fun(np.dot((Input,Weights)+bias))
def Gradient_Descent(Input, Weights, Target, Prediction, eta,bias):
    new_weight=[]
    bias=bias+eta*(Target-Prediction)
    for x,w in zip(Input,Weights):
        new_w=w+eta*(Target-Prediction)*x
        new_weight.append(new_w)
    return new_weight,bias

#DATA
Input=np.array([[0,1,0],[0,1,1],[1,1,0],[1,1,1],[1,0,0]])
Target=np.array([0,1,1,0,1])
Weights=np.array([0.3,0.1,0.5,-0.1,0.45])
bias=0.5
eta=0.01
for i in range(10):
    for x,y in zip(Input, Target):
        pred=get_prediction(x,Weights, bias)
        weights,bias=Gradient_Descent(x,Weights,y,pred,eta,bias)
```

Convolutional Neural netwok(CNN) ¶

The convolutional layer is the core building block of a CNN, and it is where the majority of computation occurs. It requires a few components, which are input data, a filter, and a feature map. Let’s assume that the input will be a color image, which is made up of a matrix of pixels in 3D. This means that the input will have three dimensions—a height, width, and depth—which correspond to RGB in an image. We also have a feature detector, also known as a kernel or a filter, which will move across the receptive fields of the image, checking if the feature is present. This process is known as a convolution.

In [1]:
```
import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Flatten
from keras.layers import Conv2D, MaxPooling2D
from keras import backend as K
import numpy as np
```

In [2]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 2s 0us/step
```

In [3]:
```
img_rows, img_cols = 28, 28

if K.image_data_format() == 'channels_first':
   x_train = x_train.reshape(x_train.shape[0], 1, img_rows, img_cols)
   x_test = x_test.reshape(x_test.shape[0], 1, img_rows, img_cols)
   input_shape = (1, img_rows, img_cols)
else:
   x_train = x_train.reshape(x_train.shape[0], img_rows, img_cols, 1)
   x_test = x_test.reshape(x_test.shape[0], img_rows, img_cols, 1)
   input_shape = (img_rows, img_cols, 1)

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

In [5]:
```
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3),
   activation = 'relu', input_shape = input_shape))
model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Dropout(0.25)) , model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation = 'softmax'))
```

In [6]:
```
model.compile(loss = keras.losses.categorical_crossentropy,
   optimizer = keras.optimizers.Adadelta(), metrics = ['accuracy'])
```

In [7]:
```
model.fit(
   x_train, y_train,
   batch_size = 128,
   epochs = 12,
   verbose = 1,
   validation_data = (x_test, y_test)
)
```

```
Epoch 1/12
469/469 [==============================] - 97s 205ms/step - loss: 2.2782 - acc
uracy: 0.1654 - val_loss: 2.2462 - val_accuracy: 0.3393
Epoch 2/12
469/469 [==============================] - 91s 193ms/step - loss: 2.2283 - acc
uracy: 0.2842 - val_loss: 2.1860 - val_accuracy: 0.5698
Epoch 3/12
469/469 [==============================] - 96s 204ms/step - loss: 2.1653 - acc
uracy: 0.3792 - val_loss: 2.1047 - val_accuracy: 0.6451
Epoch 4/12
469/469 [==============================] - 96s 205ms/step - loss: 2.0771 - acc
uracy: 0.4524 - val_loss: 1.9928 - val_accuracy: 0.6710
Epoch 5/12
469/469 [==============================] - 96s 204ms/step - loss: 1.9633 - acc
uracy: 0.5031 - val_loss: 1.8476 - val_accuracy: 0.7002
Epoch 6/12
469/469 [==============================] - 97s 208ms/step - loss: 1.8186 - acc
uracy: 0.5452 - val_loss: 1.6696 - val_accuracy: 0.7383
Epoch 7/12
469/469 [==============================] - 96s 205ms/step - loss: 1.6567 - acc
uracy: 0.5794 - val_loss: 1.4714 - val_accuracy: 0.7690
Epoch 8/12
469/469 [==============================] - 98s 210ms/step - loss: 1.4867 - acc
uracy: 0.6086 - val_loss: 1.2771 - val_accuracy: 0.7910
Epoch 9/12
469/469 [==============================] - 98s 209ms/step - loss: 1.3380 - acc
uracy: 0.6332 - val_loss: 1.1076 - val_accuracy: 0.8102
Epoch 10/12
469/469 [==============================] - 99s 211ms/step - loss: 1.2136 - acc
uracy: 0.6551 - val_loss: 0.9707 - val_accuracy: 0.8220
Epoch 11/12
469/469 [==============================] - 100s 213ms/step - loss: 1.1104 - ac
curacy: 0.6790 - val_loss: 0.8624 - val_accuracy: 0.8308
Epoch 12/12
469/469 [==============================] - 101s 214ms/step - loss: 1.0256 - ac
curacy: 0.6982 - val_loss: 0.7784 - val_accuracy: 0.8389
```

Out[7]:
```
<keras.callbacks.History at 0x21d4ce70c40>
```

In [8]:
```
score = model.evaluate(x_test, y_test, verbose = 0)

print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.7784239053726196
Test accuracy: 0.8389000296592712
```

In [9]:
```
pred = model.predict(x_test)
pred = np.argmax(pred, axis = 1)[:5]
label = np.argmax(y_test,axis = 1)[:5]

print(pred)
print(label)
```

```
313/313 [==============================] - 4s 12ms/step
[7 2 1 0 4]
[7 2 1 0 4]
```

In [ ]:
```
```