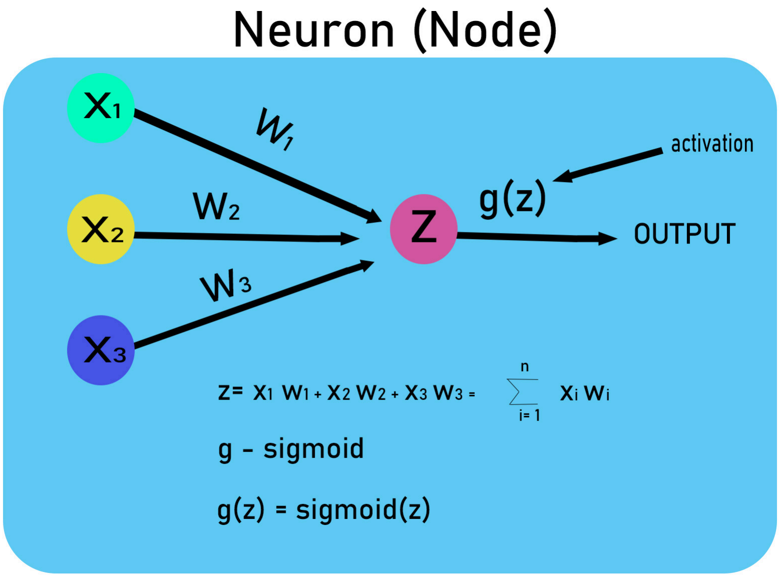
Implementation of neural network from scratch using NumPy

Last Updated : 11 Apr, 2025

Neural networks are a core component of deep learning models, and implementing them from scratch is a great way to understand their inner workings. we will demonstrate how to implement a basic Neural networks algorithm from scratch using the **NumPy** library in Python, focusing on building a three-letter classifier for the characters A, B, and C.



A neural network is a computational model inspired by the way biological neural networks process information. It consists of layers of interconnected nodes, called **neurons**, which transform input data into output. A typical neural network consists of:

- **Input Layer (X_1, X_2, X_3)** : Takes the features of the data as input.
- **Hidden Layers (Z)** : Layers between the input and output that perform transformations based on weights.
- **Output Layer**: Produces the final prediction.

Python Implementation

Step 1 : Creating the Dataset Using NumPy Arrays of 0s and 1s

As the image is a collection of pixel values in matrix, we will create a simple dataset for the letters A, B, and C using binary matrices. These matrices represent pixel values of 5x6 grids for each letter.

```
# Creating data set

# A
a =[0, 0, 1, 1, 0, 0,
   0, 1, 0, 0, 1, 0,
   1, 1, 1, 1, 1, 1,
   1, 0, 0, 0, 0, 1,
   1, 0, 0, 0, 0, 1]
# B
b =[0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 1, 0,
   0, 1, 1, 1, 1, 0]
# C
c =[0, 1, 1, 1, 1, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 0, 0, 0, 0,
   0, 1, 1, 1, 1, 0]

# Creating labels
y =[[1, 0, 0],
   [0, 1, 0],
   [0, 0, 1]]
```

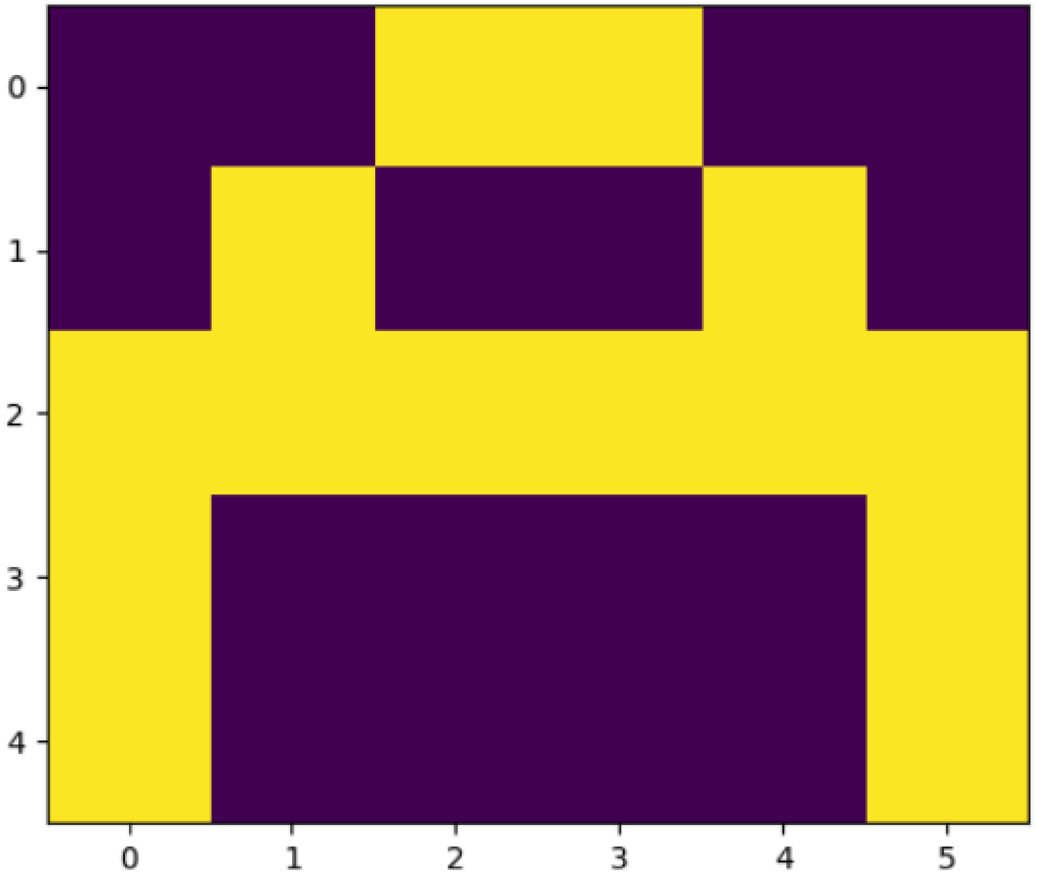
Step 2 : Visualizing the Dataset

To visualize the datasets, we can use **Matplotlib** to plot the images for each letter. This will give us a clear understanding of what the data looks like before feeding it into the neural network.

```
import numpy as np
import matplotlib.pyplot as plt

# visualizing the data, plotting A.
plt.imshow(np.array(a).reshape(5, 6))
plt.show()
```

Output



Visualizing the Data

Step 3 : As the data set is in the form of list we will convert it into numpy array.

We convert the lists of pixel values and the corresponding labels into **NumPy arrays** to work with them efficiently in the neural network.

```
# converting data and labels into numpy array
x =[np.array(a).reshape(1, 30), np.array(b).reshape(1, 30),
                                np.array(c).reshape(1, 30)]

y = np.array(y)
# Printing data and labels
```

```
print(x, "\n\n", y)
```

Output

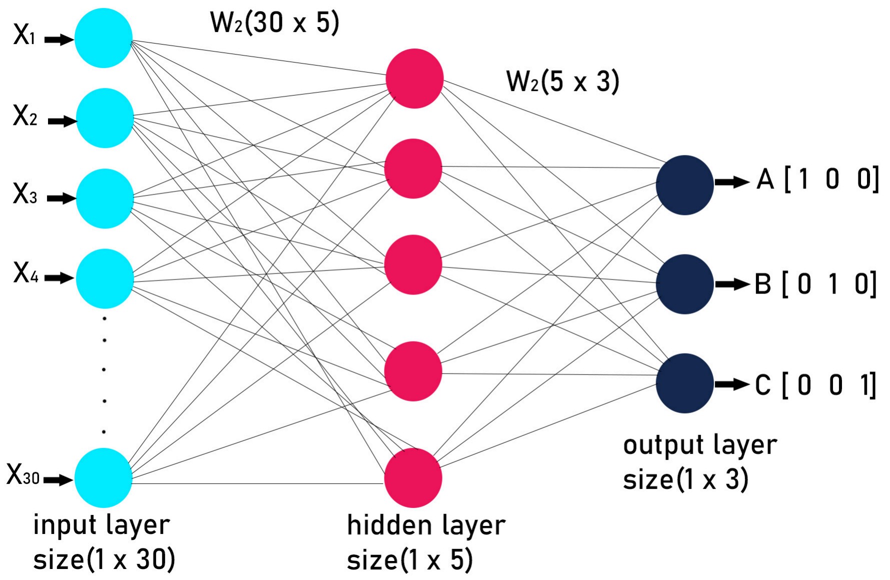
```
[array([[0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1,
1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1]]),
 array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0,
0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0]]),
 array([[0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 1, 1, 1,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

[[1 0 0]
[0 1 0]
[0 0 1]]
```

Step 4: Defining the Architecture of the Neural Network

Our neural network will have the following structure:

- **Input Layer**: 1 layer with 30 nodes (representing the 5x6 grid).
- **Hidden Layer**: 1 layer with 5 nodes.
- **Output Layer**: 1 layer with 3 nodes (representing the letters A, B, and C).



Step 5: Defining the Neural Network Functions

Here, we will define the key components of the neural network:

- **Activation Function**: We'll use the **sigmoid** activation function.
- **Feedforward Process**: Computes the output by passing the input through the layers.
- **Backpropagation**: Updates weights to minimize the loss.
- **Loss Function**: We’ll use **Mean Squared Error (MSE)** to compute the loss.

```
# activation function
def sigmoid(x):
    return(1/(1 + np.exp(-x)))

# Creating the Feed forward neural network
def f_forward(x, w1, w2):
    # hidden
    z1 = x.dot(w1)     # input from layer 1
    a1 = sigmoid(z1)  # out put of layer 2
    z2 = a1.dot(w2)   # input of out layer
    a2 = sigmoid(z2)  # output of out layer
    return(a2)

# initializing the weights randomly
def generate_wt(x, y):
    li =[]
    for i in range(x * y):
        li.append(np.random.randn())
    return(np.array(li).reshape(x, y))

# for loss we will be using mean square error(MSE)
def loss(out, Y):
    s =(np.square(out-Y))
    s = np.sum(s)/len(y)
    return(s)

# Back propagation of error
def back_prop(x, y, w1, w2, alpha):

    # hidden layer
    z1 = x.dot(w1)
    a1 = sigmoid(z1)
    z2 = a1.dot(w2)
    a2 = sigmoid(z2)
    # error in output layer
    d2 =(a2-y)
    d1 = np.multiply((w2.dot((d2.transpose()))).transpose(),
                                (np.multiply(a1, 1-a1)))
    # Gradient for w1 and w2
    w1_adj = x.transpose().dot(d1)
    w2_adj = a1.transpose().dot(d2)

    # Updating parameters
    w1 = w1-(alpha*(w1_adj))
    w2 = w2-(alpha*(w2_adj))

    return(w1, w2)
```

Step 6: Initializing Weights

We initialize the weights for both the hidden layer and the output layer randomly.

```
w1 = generate_wt(30, 5)
w2 = generate_wt(5, 3)

print(w1, "\n\n", w2)
```

Output

```
[[ 0.75696605 -0.15959223 -1.43034587  0.17885107 -0.75859483]
 [-0.22870119  1.05882236 -0.15880572  0.11692122  0.58621482]
 [ 0.13926738  0.72963505  0.36050426  0.79866465 -0.17471235]
 [ 1.00708386  0.68803291  0.14110839 -0.7162728   0.69990794]
 [-0.90437131  0.63977434 -0.43317212  0.67134205 -0.9316605 ]
 [ 0.15860963 -1.17967773 -0.70747245  0.22870289  0.00940404]
 [ 1.40511247 -1.29543461  1.41613069 -0.97964787 -2.86220777]
 [ 0.66293564 -1.94013093 -0.78189238  1.44904122 -1.81131482]
 [ 0.4441061  -0.18751726 -2.58252033  0.23076863  0.12182448]
 [-0.60061323  0.39855851 -0.55612255  2.0201934   0.70525187]
 [-1.82925367  1.32004437  0.03226202 -0.79073523 -0.20750692]
 [-0.25756077 -1.37543232 -0.71369897 -0.13556156 -0.34918718]
 [ 0.26048374  2.49871398  1.01139237 -1.73242425 -0.67235417]
 [ 0.30351062 -0.45425039 -0.84046541 -0.60435352 -0.06281934]
 [ 0.43562048  0.66297676  1.76386981 -1.11794675  2.2012095 ]
 [-1.11051533  0.3462945   0.19136933  0.19717914 -1.78323674]
 [ 1.1219638  -0.04282422 -0.0142484  -0.73210071 -0.58364205]
 [-1.24046375  0.23368434  0.62323707 -1.66265946 -0.87481714]
 [ 0.19484897  0.12629217 -1.01575241 -0.47028007 -0.58278292]
 [ 0.16703418 -0.50993283 -0.90036661  2.33584006  0.96395524]
 [-0.72714199  0.39000914 -1.3215123   0.92744032 -1.44239943]
 [-2.30234278 -0.52677889 -0.09759073 -0.63982215 -0.51416013]
 [ 1.25338899 -0.58950956 -0.86009159 -0.7752274   2.24655146]
 [ 0.07553743 -1.2292084   0.46184872 -0.56390328  0.15901276]
 [-0.52090565 -2.42754589 -0.78354152 -0.44405857  1.16228247]
 [-1.21805132 -0.40358444 -0.65942185  0.76753095 -0.19664978]
 [-1.5866041   1.17100962 -1.50840821 -0.61750557  1.56003127]
 [ 1.33045269 -0.85811272  1.88869376  0.79491455 -0.96199293]
 [-2.34456987  0.1005953  -0.99376025 -0.94402235 -0.3078695 ]
 [ 0.93611909  0.58522915 -0.15553566 -1.03352997 -2.7210093 ]]

 [[-0.50650286 -0.41168428 -0.7107231 ]
 [ 1.86861492 -0.36446849  0.97721539]
 [-0.12792125  0.69578056 -0.6639736 ]
 [ 0.58190462 -0.98941614  0.40932723]
 [ 0.89758789 -0.49250365 -0.05023684]]
```

Step 7: Training the Model

Now that we've defined the structure, functions, and initialized the weights, we can train the model using the **train** function. This function will update the weights through backpropagation for a specified number of epochs.

```
def train(x, Y, w1, w2, alpha = 0.01, epoch = 10):
    acc =[]
    losss =[]
    for j in range(epoch):
        l =[]
        for i in range(len(x)):
            out = f_forward(x[i], w1, w2)
            l.append((loss(out, Y[i])))
            w1, w2 = back_prop(x[i], y[i], w1, w2, alpha)
        print("epochs:", j + 1, "======== acc:", (1-
(sum(l)/len(x)))*100)
        acc.append((1-(sum(l)/len(x)))*100)
        losss.append(sum(l)/len(x))
    return(acc, losss, w1, w2)

acc, losss, w1, w2 = train(x, y, w1, w2, 0.1, 100)
```

Step 8: Plotting Accuracy and Loss

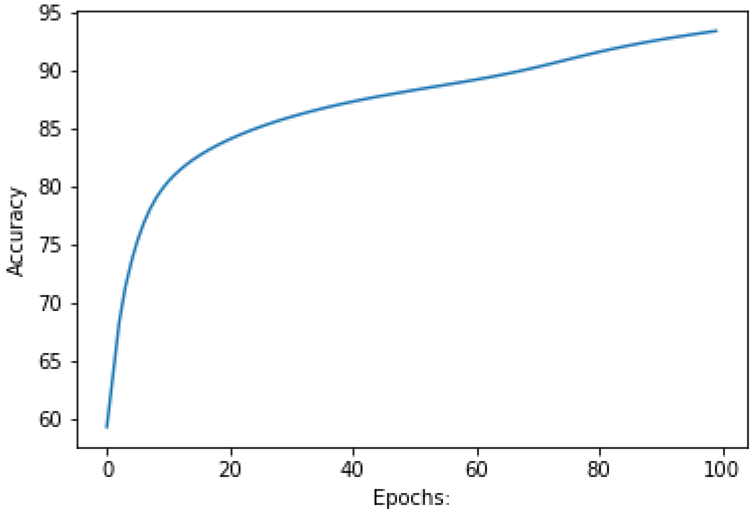
After training, we can visualize the **accuracy** and **loss** over the epochs to understand the model's learning process.

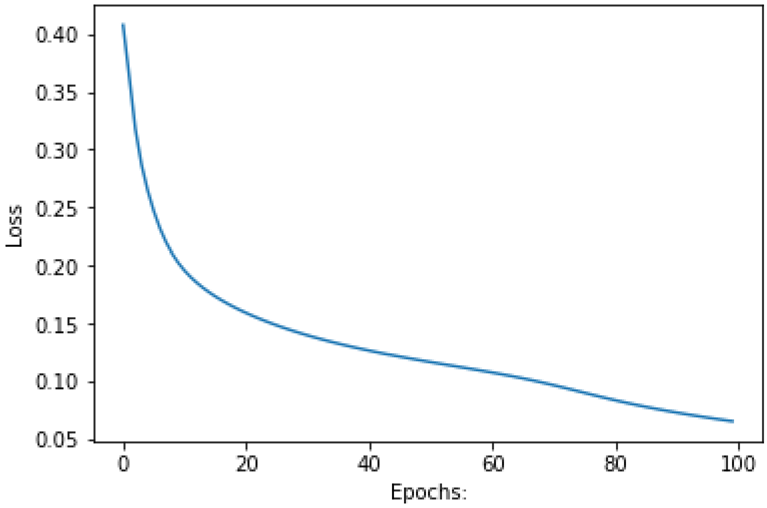
```
import matplotlib.pyplot as plt1

# plotting accuracy
plt1.plot(acc)
plt1.ylabel('Accuracy')
plt1.xlabel("Epochs:")
plt1.show()

# plotting Loss
plt1.plot(losss)
plt1.ylabel('Loss')
plt1.xlabel("Epochs:")
plt1.show()
```

Output





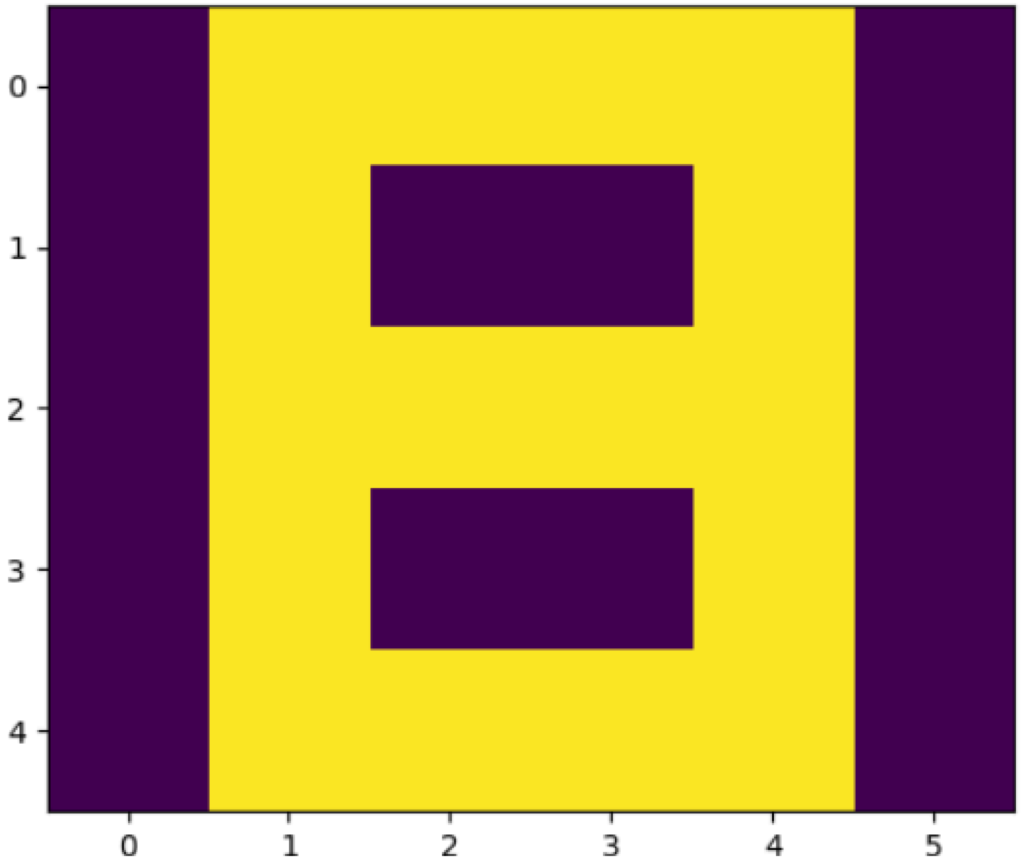
Step 9: Making Predictions

We use the trained weights to predict the letter class for a new input. The class with the highest output value is chosen as the predicted class.

```
def predict(x, w1, w2):
    Out = f_forward(x, w1, w2)
    maxm = 0
    k = 0
    for i in range(len(Out[0])):
        if(maxm<Out[0][i]):
            maxm = Out[0][i]
            k = i
    if(k == 0):
        print("Image is of letter A.")
    elif(k == 1):
        print("Image is of letter B.")
    else:
        print("Image is of letter C.")
    plt.imshow(x.reshape(5, 6))
    plt.show()
# Example: Predicting for letter 'B'
predict(x[1], w1, w2)
```

Output

```
Image is of letter B.
```



Predicted Image

Comment

More info

Explore

NumPy Tutorial - Python Library 3 min read

Introduction

Creating NumPy Array

NumPy Array Manipulation

Matrix in NumPy

Operations on NumPy Array

Reshaping NumPy Array

Indexing NumPy Array

Arithmetic operations on NumPyArray

Linear Algebra in NumPy Array

GeeksforGeeks
Sanchhaya Education Private Limited

Corporate & Communications Address:

A-143, 7th Floor, Sovereign Corporate
Tower, Sector- 136, Noida, Uttar Pradesh
(201305)

Registered Address:

K 061, Tower K, Gulshan Vivante
Apartment, Sector 137, Noida, Gautam
Buddh Nagar, Uttar Pradesh, 201305

Company

About Us
Legal
Privacy Policy
Contact Us
Advertise with us
GFG Corporate Solution
Campus Training Program

Explore

POTD
Job-A-Thon
Community
Blogs
Nation Skill Up

Tutorials

Programming Languages
DSA
Web Technology
AI, ML & Data Science
DevOps
CS Core Subjects
Interview Preparation
GATE
Software and Tools

Courses

IBM Certification
DSA and Placements
Web Development
Programming Languages
DevOps & Cloud
GATE
Trending Technologies

Videos

Preparation Corner