Creating a Neural Network from scratch

Course Instructor : Khaled Mostafa El Sayed
TAs : Eng. Gamal Zayed & Eng. Marwa Monier

Sama Youssef 202-000-819

December 12th, 2023

1 Abstract

This paper provides an overview of neural networks, a class of algorithms inspired by the structure and function of the human brain. The abstract begins with an introduction to the fundamental concepts of neural networks, their historical development, and their contemporary applications. Furthermore, it discusses the basic structure and functionality of neural networks, along with their ability to learn and adapt from data. The abstract also highlights some prominent types of neural networks and their specific uses in various domains. Finally, it enumerates the potential future advancements and challenges in the field of neural networks.

2 Introduction

The use of artificial intelligence (AI) has gained significant traction in recent years, with neural networks serving as one of the primary tools for powering AI applications. The introduction provides a brief historical background on the development of neural networks, tracing their origins and early successes. Additionally, it delineates the motivation for using neural networks, emphasizing their capability to solve complex problems that are challenging for traditional algorithmic approaches. The introduction also outlines the structure and function of neural networks, showcasing their ability to recognize patterns, make predictions, and perform various cognitive tasks. Lastly, it sets the stage for delving deeper into the different types of neural networks and their respective contributions to the ever-expanding landscape of AI and machine learning.

3 Initializing the weights(w)

The function is designed to create and initialize the weights (w) for a neural network with multiple layers. The function takes four parameters: the total number of layers in the neural network including input and output layers), the number of neurons in the input layer, the number of neurons in the hidden layers, and the number of neurons in the output layer.

```
def generate_w(num_layer, input_size,num_nodes,output_size):
  input_layer = [[0 for _ in range(input_size)] for _ in range(num_nodes)]
  input_layer2 = [[0 for _ in range(num_nodes)] for _ in range(num_nodes)]
  w=[]
  w.append(input_layer)
  for i in range (num_layer-3):
    w.append(input_layer2)

  outputlayer = [[0 for _ in range(num_nodes)] for _ in range(output_size)]
  w.append(outputlayer)
  return w
```

4 Initializing the neurons for a neural network

The function is intended to create and initialize the nodes or neurons for a neural network with multiple layers. The function takes four parameters: the number of neurons in the hidden layers, the number of neurons in the output layer, the number of neurons in the input layer, and the total number of layers in the neural network including input and output layers.

```
def generate_Nodes(hiden_layer_size,output_layer_size,input_sample_size,num_of_layers):

  Nodes=[]
  input_layer = [[0 for _ in range(3)] for _ in range(input_sample_size)]
  Nodes.append(input_layer)
  for i in range (num_of_layers-2):
    hiden_layer=[[0 for _ in range(3)] for _ in range(hiden_layer_size)]
    Nodes.append(hiden_layer)


  output_layer=[[0 for _ in range(3)] for _ in range(output_layer_size)]
  Nodes.append(output_layer)

  return Nodes
```

5 sigmoid activation function

The "sigmoid" function is a mathematical function used in machine learning and neural networks to introduce non-linearity into the output of a neuron. It takes an input value and applies the sigmoid activation function to it. The sigmoid function is defined as 1 / (1 + exp(-x)), where exp(-x) represents the exponential function. When the input is passed through this function, it compresses the output to a range between 0 and 1, which can be interpreted as a probability. This function is commonly used to simulate the firing of a neuron in a neural network and is especially useful in binary classification tasks.

```
def sigmoid(input):
  return 1 / (1 + np.exp(-input))
```

6 Forward propagation

The "forward" function is intended to perform the forward propagation process in a neural network. This function takes two parameters: "W" (weights) and "Nodes" (neuron values).this function effectively performs the forward propagation by updating the neuron values in each layer of the neural network based on the given weights and neuron values.

```
def forward(W,Nodes):

  rows = len(Nodes)       #
  cols = len(Nodes[0])    #3

  for row in range (len(Nodes)-1) :

    for nod in range(len(Nodes[row+1])):

      for i in range(len(Nodes[row])):

        Nodes[row+1][nod][0]=Nodes[row+1][nod][0]+Nodes[row][i][1]*W[row][nod][i]
      Nodes[row+1][nod][1]=sigmoid(Nodes[row+1][nod][0])

  return Nodes
```

7 Back propagation

This function conducts the back propagation process in a neural network. It takes the nodes, weights, target values, and the learning rate (eta) as input.the function iterates through the network layers in reverse order, calculating the error terms (sigma) and updating the weights based on the error using the specified learning rate (eta). The updated weights are then returned.

```

def backword(nodes,w,target,eta):
  rows = len(nodes)
  for row in range(rows-1, -1, -1):
    for nod in range(len(nodes[row])):
      if row ==rows-1:
          segma=nodes[row][nod][1]*(1-nodes[row][nod][1])*(target[nod]-nodes[row][nod
    ][1])
          nodes[row][nod][2]=segma
          if row!=0:
            for i in range(len(nodes[row-1])):
              delta_w=eta*nodes[row-1][i][1]*segma
              w[row-1][nod][i]=w[row-1][nod][i]+delta_w

      else:
        for k in range(len(nodes[row+1])):
          sum=nodes[row+1][k][2]*w[row][k][nod]
        segma=nodes[row][nod][1]*(1-nodes[row][nod][1])*sum
        nodes[row][nod][2]=segma
        if row!=0:
          for j in range(len(nodes[row-1])):
            delta_w=eta*nodes[row-1][j][1]*segma
            w[row-1][nod][j]=w[row-1][nod][j]+delta_w

```

```
24
25    return(w)
```

8 Loading Iris dataset

this code loads the Iris dataset, prepares the features and target variable for training and testing, converts the data to lists, and assigns the lists to input list and target list for further use.

```
iris = datasets.load_iris()
iris_X = pd.DataFrame(iris.data, columns=iris.feature_names)
iris_Y = pd.get_dummies(iris.target)
X_train, X_test, Y_train, Y_test = train_test_split(iris_X, iris_Y, test_size=0.2,
    random_state=42)
iris_X_list=iris_X.values.tolist()
iris_Y_list=iris_Y.values.tolist()
input_list=iris_X_list
target_list=iris_Y_list
```

9 Loading MNIST dataset

this code loads the MNIST dataset, prepares the features and target variable for training and testing, converts the data to lists, and assigns the lists to input list and target list for further use. This is useful for preparing the MNIST data for training a machine learning model.

```
mnist = fetch_openml('mnist_784')

# Convert to DataFrame and normalize the pixel values
mnist_X = pd.DataFrame(mnist.data / 255.0)
mnist_Y = pd.get_dummies(mnist.target)

# Split the data into training and testing sets
X_train_mnist, X_test_mnist, Y_train_mnist, Y_test_mnist = train_test_split(mnist_X,
    mnist_Y, test_size=0.001, random_state=42)
mnist_X_list = X_test_mnist.values.tolist()
mnist_Y_list = Y_test_mnist.values.tolist()
input_list=mnist_X_list
target_list=mnist_Y_list
```

10 Collecting the predicted outputs

this a loop that iterates through input samples, performs forward and backward propagation to train the neural network model, and collects the predicted output values.

```

num_of_nodes=3
eta=3
num_layer=3
yred_list=[]

w=generate_w(num_layer, len(input_list[0]),num_of_nodes,len(target_list[0]))

for j in range(len(input_list)):

  nodes=generate_Nodes(num_of_nodes,len(target_list[0]),len(input_list[0]),num_layer)

  input_sample =input_list[j]
  target=target_list[j]
  for k in range(len(nodes[0])):
     nodes[0][k][1]=input_sample[k]

  #print(nodes)
  for i in range(10):
    nodes=forward(w,nodes)
    backword(nodes,w,target,eta)
  y_list=[]

  for nod in range(len(nodes[num_layer-1])):
```

```
    yred=nodes[2][nod][1]
    y_list.append(yred)
  yred_list.append(y_list)
```

11 Accuracy score

This code snippet essentially prepares the predicted values for evaluation by rounding them to integers, creates a DataFrame to organize the predictions, and calculates the accuracy score. The accuracy score is a measure of the model's performance in predicting the test data. The printed accuracy score provides an indicator of how well the model is performing.

```
columns = [f"col{i}" for i in range(1,len(yred_list[0])+1 , 1)]
print(columns)
rounded_pred_list = [[int(round(value)) for value in row] for row in yred_list]
y_pred = pd.DataFrame(rounded_pred_list, columns=columns)
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(Y_test_mnist, y_pred)
print("Accuracy Score:", accuracy*100,"%")
```

12 Results

this is the accuracy of Iris dataset with 20 iteration, learning rate 5, number of nodes = 4 and number of layers= 3

Figure 1:

the accuracy of mnsit dataset with 20 iteration, learning rate 5, number of nodes = 4 and number of layers= 3

Figure 2: