

Department of Information Engineering Technology
The Superior University, Lahore

Artificial Intelligence Lab

Experiment No.9

Implementation of Neural Network in Python from Scratch

(Part -1)

Prepared for

By:

Name: _____

ID: _____

Section: _____

Semester: _____

Total Marks: _____

Obtained Marks: _____

Signature: _____

Date: _____

Experiment No. 9

Implementation of Neural Network in Python from Scratch (Part -1)

(Rubrics)

Name: _____

Roll No. : _____

A. PSYCHOMOTOR

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Follow Procedures	1	0	0.25	0.5	0.75	1	
2	Software and Simulations	2	0	0.5	1	1.5	2	
3	Accuracy in Output Results	3	0	0.75	1.5	2.25	3	
Sub Total		6	Sub Total marks Obtained in Psychomotor(P)					

B. AFFECTIVE

Sr. No.	Criteria	Allocated Marks	Unacceptable 0%	Poor 25%	Fair 50%	Good 75%	Excellent 100%	Total Obtained
1	Respond to Questions	1	0	0.25	0.5	0.75	1	
2	Lab Report	1	0	0.25	0.5	0.75	1	
3	Assigned Task	2	0	0.5	1	1.5	2	
Sub Total		4	Sub Total marks Obtained in Affective (A)					

Instructor Name: _____

Total Marks (P+A): _____

Instructor Signature: _____

Obtained Marks (P+A): _____

Introduction to Activation Function

An activation function is a mathematical operation applied to the output of each neuron (or node) in a neural network layer. It introduces non-linearity to the network, allowing it to learn and approximate complex, non-linear relationships in the data. The purpose of the activation function is to determine the output of a neural network node, given an input or set of inputs.

In a neural network, each neuron receives inputs, performs a weighted sum of those inputs, adds a bias, and then passes the result through an activation function. This transformed output is then passed as input to the next layer of neurons in the network. The activation function serves as a decision-making element, determining whether a neuron should be activated (produce an output) or not.

Without activation functions (or with linear activation functions), a neural network would behave like a linear model, no matter how many layers it has. The introduction of non-linear activation functions enables the network to learn complex patterns and relationships in the data, making it capable of approximating a wide range of functions.

Common activation functions include sigmoid, hyperbolic tangent (tanh), rectified linear unit (ReLU), and their variants. The choice of activation function depends on the specific requirements of the task and the characteristics of the data being processed. Different activation functions have different properties and are suitable for different types of problems.

Non-Linearity:

One of the primary purposes of activation functions is to introduce non-linearity into the network. This non-linearity is crucial for the network to learn and represent complex relationships in the data. Without non-linear activation functions, no matter how deep the network is, it would essentially behave like a linear model, and its expressive power would be limited.

Activation Threshold:

Activation functions introduce an activation threshold, which determines the point at which a neuron "fires" or becomes activated. Neurons only produce an output when the weighted sum of their inputs exceeds this threshold. This binary activation/non-activation property is essential for the network's ability to model intricate decision boundaries.

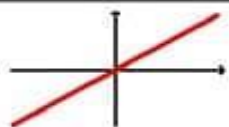
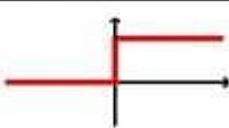
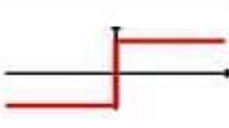


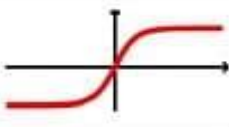

Differentiability:

Many optimization algorithms, such as gradient descent, rely on the derivatives of activation functions. Hence, it's beneficial for activation functions to be differentiable. This property allows the network to learn through backpropagation, adjusting weights based on the gradient of the loss function with respect to the network parameters.

Vanishing and Exploding Gradients:

Activation functions can help mitigate issues like vanishing and exploding gradients during training. Vanishing gradients occur when the gradient becomes very small, leading to slow or stalled learning. Exploding gradients happen when the gradient becomes too large, causing the model parameters to update dramatically. Well-designed activation functions can help stabilize and facilitate the training process.

Here are a few commonly used activation functions:

Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \leq -1/2 \\ z + 1/2 & -1/2 \leq z \leq 1/2 \\ 1 & z \geq 1/2 \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multilayer NN	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$	Multilayer NN, RNNs	
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$	Multilayer NN, CNNs	

Activation Function with Python

Here's how Activation Function works:

Sigmoid Function:

The sigmoid function is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$ and code correctly follows this formula using the `math.exp` function to compute the exponentiation, with input values to the range (0, 1).

```
import math
def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

```
sigmoid(100) # output 1.0
```

Tanh Function:

Implementing hyperbolic tangent (tanh) function using the formula $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ looks correct. This is the standard formula for the tanh function, which squashes input values to the range (-1, 1).

```
import math
def tanh(x):
    return (math.exp(x) - math.exp(-x)) / (math.exp(x) + math.exp(-x))
tanh(100) # output 1.0
```

ReLU Function:

The ReLU activation function is defined as $ReLU(x) = \max(0, x)$ with range $(0, \infty)$

```
def relu(x):
    return max(0, x)
relu(-100) # output 0
```

Calculate Profit/Loss from Revenue and Expenses

To calculate the profit by subtracting the expenses array from the revenue array.

This code will output the profit matrix, where each element is the result of subtracting the corresponding element in the expenses matrix from the revenue.

```
import numpy as np
revenue = np.array([[180, 200, 220], [24, 36, 40], [12, 18, 20]])
expenses = np.array([[80, 90, 100], [10, 16, 20], [8, 10, 10]])
Profit = revenue - expenses
Profit
```

Calculate Total Sales from Units and Price per Unit using Matrix Multiplication

This code uses the dot product of the price_per_unit array and the units array to calculate the total sales. The result is a 1D array representing the total sales for each entity or category.

```
import numpy as np
price_per_unit = np.array([1000, 400, 1200])
units = np.array([[30, 40, 50], [5, 10, 15], [2, 5, 7]])
price_per_unit * units
```

Introduction to Loss Function

In the context of machine learning, a loss function (or cost function) is a mathematical function that quantifies how well a machine learning model is performing with respect to its training data. The purpose of a loss function is to measure the difference between the predicted output of the model and the actual target values (ground truth) for a given set of input data. The goal during the training process is to minimize this loss function.

Here are key points regarding loss functions and their significance:

Quantifying Model Performance:

The loss function provides a numerical measure of how well the model is predicting the target values. It quantifies the error or deviation between the predicted values and the actual values.

Training Objective:

During the training phase, the machine learning algorithm iteratively adjusts the model's parameters (weights and biases) to minimize the loss function. This optimization process is typically performed using techniques like gradient descent.

Guiding Learning:

Minimizing the loss function guides the learning process by updating the model parameters in a way that reduces prediction errors. The model aims to find the set of parameters that minimizes the overall loss across the entire training dataset.

Different Tasks, Different Loss Functions:

The choice of the loss function depends on the specific task the model is designed to perform. Different tasks (e.g., regression, binary classification, multiclass classification) require different types of loss functions that align with the nature of the output and the desired behavior of the model.

Evaluation Metric vs. Loss Function:

While the loss function is used for training the model, evaluation metrics (such as accuracy, precision, recall) are used to assess the model's performance on validation or test data. The loss function focuses on the optimization process, while evaluation metrics provide a more interpretable measure of performance.

Examples of Loss Functions:

- **Mean Squared Error (MSE):** Commonly used for regression tasks.
- **Binary Crossentropy:** Used for binary classification tasks.
- **Categorical Crossentropy:** Used for multiclass classification tasks.
- **Hinge Loss:** Used in support vector machines (SVMs).
- **Huber Loss:** Used in regression tasks, particularly when robustness to outliers is desired.

In summary, loss functions play a crucial role in the training of machine learning models by providing a quantitative measure of how well the model is learning from the data. The goal is to find the model parameters that minimize this measure of error, resulting in a model that generalizes well to unseen data.

Loss Function with Python

Here's how Loss Function works:

Imports the NumPy library and gives it the alias np. NumPy is a powerful library for numerical operations in Python, and it provides efficient and convenient functions for working with arrays.

```
import numpy as np
```

Creating NumPy arrays y_predicted and y_true representing the predicted and true values, respectively.

```
y_predicted = np.array([1,1,0,0,1])
y_true = np.array([0.30,0.7,1,0,0.5])
```

Implement Mean Absolute Error

- Define a function mae that takes y_predicted and y_true as inputs.
- Use a loop to iterate through each pair of predicted and true values.
- Calculate the absolute difference between each predicted and true value and accumulate the total error.
- Print the total error and mean absolute error.
- Return the mean absolute error.

```
def mae(y_predicted, y_true):
    total_error = 0
    for yp, yt in zip(y_predicted, y_true):
        total_error += abs(yp - yt)
    print("Total error is:", total_error)
    mae = total_error/len(y_predicted)
    print("Mean absolute error is:", mae)
    return mae
mae(y_predicted, y_true)
```

Implement same thing using numpy in much easier way

- Use NumPy's element-wise subtraction (y_predicted - y_true) to get an array of absolute differences.
- Use np.abs to compute the absolute values of the differences.
- Use np.mean to calculate the mean of the absolute differences.
- Define a function (mae_np) that encapsulates this NumPy-based calculation.
- Call the mae_np function with y_predicted and y_true to obtain the mean absolute error.

```
np.abs(y_predicted-y_true)
np.mean(np.abs(y_predicted-y_true))
```

```
def mae_np(y_predicted, y_true):
    return np.mean(np.abs(y_predicted-y_true))
mae_np(y_predicted, y_true)
```

Implement Log Loss or Binary Cross Entropy

Epsilon typically refers to a very small positive number. It is used to avoid numerical instability or undefined behavior when certain mathematical operations involve very small or zero values.

- Define a small epsilon value (1e-15)
- Create a new list (y_predicted_new) where each element is the maximum of the original predicted value and epsilon. This step is taken to avoid taking the log of zero in the next calculations
- Calculate the value of 1-epsilon, which is used in the context of log loss calculations.

```
np.log([0])
epsilon = 1e-15
np.log([1e-15])
y_predicted
y_predicted_new = [max(i,epsilon) for i in y_predicted]
y_predicted_new
1-epsilon
```

Lab Task:

1. Below are some Indian company's revenues in US dollars. Using NumPy can you convert this into Indian rupees? 1 USD = 75 INR

	Revenue(Million US \$)		
	FY 2018	FY 2019	FY 2020
Reliance Industries	200	220	250
Infosys	68	79	105
TCS	110	140	180
Wipro	80	85	90

2. Divine flowers are flower shop that sells different type of flowers. Below is the table showing how many flowers of each type they sold in different months. Also given are the prices of one flower each. Using this find out their total sales in every month.

	Units Sold		
	Jan	Feb	March
Roses	50	60	25
Lotus	10	13	5
Sunflower	40	70	52

Price per unit	
Roses	20
Lotus	30
Sunflower	15

Home Task:

1. Create Tanh, ReLU and Leaky ReLU activation functions by their formulas.
2. Make Categorical Cross-Entropy and Mean Square Error loss function.