Practical No. 3

Title: Implementation of Artificial Neural Network algorithms

Aim: To implement and evaluate Artificial Neural Network (ANN) algorithms for solving classification and regression problems using Python

Introduction:

Artificial Neural Networks (ANNs) are computational models inspired by the structure and functioning of the human brain. They consist of interconnected processing elements called neurons that work collectively to solve specific problems. ANNs are widely used in various applications such as classification, regression, pattern recognition, and signal processing due to their ability to learn from data and generalize well to unseen inputs.

Two foundational algorithms in the field of neural networks are the Perceptron and ADALINE (Adaptive Linear Neuron). These models are considered the building blocks of more complex neural networks and are essential for understanding the fundamentals of supervised learning.

- Perceptron Algorithm:
  The Perceptron is one of the earliest neural network models, introduced by Frank Rosenblatt in 1958. It is a binary classifier that updates its weights based on the prediction error using a simple update rule. It works well for linearly separable datasets and uses a step activation function.

- ADALINE Algorithm:
  ADALINE, developed by Bernard Widrow and Ted Hoff, is similar to the perceptron but differs in the way it updates weights. Instead of using a binary output, ADALINE uses a linear activation function and minimizes the mean squared error (MSE) between predicted and actual outputs. This allows smoother and more stable convergence during training.

Both algorithms utilize the Gradient Descent optimization technique to minimize the error. Gradient Descent updates the weights iteratively in the direction of the negative gradient of the loss function to reach the optimal solution.

In this practical, we implement the Perceptron and ADALINE algorithms and explore how they learn from training data through weight updates using Gradient Descent. This provides a foundational understanding of how learning occurs in neural networks.

Exercise -

1. Implement Perceptron algorithm for neural networks
2. Implement Perceptron algorithm for OR operation
3. Implement Perceptron algorithm for AND operation

Implementation:

Program: Implement Perceptron algorithm for neural networks

```
#implementing-the-perceptron-neural-network-with-python

x_input=[0.1,0.5,0.2]

w_weights = [0.4,0.3,0.6]

threshold = 0.5


def step(weighted_sum):

 if weighted_sum>threshold:

  return 1

 else:

  return 0
```

```
def perceptron():
 weighted_sum=0


 for x,w in zip(x_input,w_weights):
  weighted_sum += x*w
  print(weighted_sum)
 return step(weighted_sum)


output=perceptron()
print("output" + str(output))
```

Output:

```
0.04000000000000001
0.19
0.31
output0
```

Program: Implement Perceptron algorithm for OR operation

```
import numpy as np
class Perceptron:
  def __init__(self, learning_rate=0.01, n_iters=1000):
    self.lr = learning_rate
    self.n_iters = n_iters
    self.activation_func = self._unit_step_func
    self.weights = None

    self.bias = None


def fit(self, X, y):

    n_samples, n_features = X.shape

    # init parameters

    self.weights = np.zeros(n_features)

    self.bias = 0

    y_ = np.array([1 if i > 0 else 0 for i in y])

    for _ in range(self.n_iters):

        for idx, x_i in enumerate(X):

            linear_output = np.dot(x_i, self.weights) + self.bias

            y_predicted = self.activation_func(linear_output)

            # Perceptron update rule

            update = self.lr * (y_[idx] - y_predicted)

            self.weights += update * x_i

            self.bias += update

def predict(self, X):

    linear_output = np.dot(X, self.weights) + self.bias

    y_predicted = self.activation_func(linear_output)

    return y_predicted
```

```
  def _unit_step_func(self, x):
      return np.where(x >= 0, 1, 0)
# OR gate inputs and outputs
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y = np.array([0, 1, 1, 1])
# Initialize and train the perceptron
perceptron = Perceptron(learning_rate=0.1, n_iters=10)
perceptron.fit(X, y)
# Test the perceptron
predictions = perceptron.predict(X)
predictions
```

Output:

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

Program: Implement Perceptron algorithm for AND operation

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import SGDClassifier
from sklearn.preprocessing import StandardScaler

# Define AND operation dataset
```

```
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])  # Input features

y = np.array([0, 0, 0, 1])  # AND operation labels


# Standardize the data (important for Adaline)

scaler = StandardScaler()

X_scaled = scaler.fit_transform(X)


# Implement Adaline using scikit-learn's SGDClassifier with linear loss

adaline = SGDClassifier(loss="squared_loss", learning_rate="constant", eta0=0.01,
max_iter=1000)

adaline.fit(X_scaled, y)


# Predict on training data

predictions = adaline.predict(X_scaled)


# Display results

print("Predictions:", predictions)

print("Actual Labels:", y)
```