Exercício 1

Matheus Arataque Uema - 10276949

SCC5809 - Redes Neurais e Aprendizado Profundo

1 Exercício

O objetivo desse exercício era implementar e treinar o modelo Adaline para reconhecer os símbolos Y e Y invertido. Primeiramente, o modelo Adaline (Adaptive Linear Element) criado por Widrow e Hoff em 1960, possui estrutura similar a de um Perceptron, diferenciando-se no algoritmo de treinamento o qual utiliza-se do método dos mínimos quadrados.

Para sua implementação, desenvolveu-se um código em *Python*[1], com o auxílio da biblioteca *NumPy*[2] para computação científica. O desenvolvimento foi dividido em três etapas:

- Criação dos dados de treino
- Implementação e treinamento do modelo
- Criação dos dados de teste e predição com modelo treinado

Duas variáveis foram declaradas para referirem-se aos símbolos de Y e Y invertido de forma que cada variável possui o seu rótulo juntamente com a matrix correspondente, como pode ser visualizado no Listing 1. Para criar a base de treino, as matrizes foram transformadas em vetores simples pelo método *flatten*, e dividiu-se a base em X_train e y_train, onde estão contidos os vetores dos dados e dos rótulos, respectivamente.

Listing 1: Declaração das variáveis

```
# "Y" Symbol with label 1
y_symbol = (1, np.array([
    [ 1, -1, -1, -1,  1],
    [ 1, -1, -1, -1,  1],
    [-1,  1,  1,  1, -1],
    [-1, -1,  1, -1, -1],
    [-1, -1,  1, -1, -1]
]))
```

[1]Python.org Disponível em: https://www.python.org/. Acesso em: 27 de ago. de 2024.
[2]NumPy Disponível em: https://numpy.org/. Acesso em: 27 de ago. de 2024.

```
# Inverted "Y" Symbol with label -1
y_inverted_symbol = (-1, np.array([
    [-1, -1,  1, -1, -1],
    [-1, -1,  1, -1, -1],
    [-1,  1,  1,  1, -1],
    [ 1, -1, -1, -1,  1],
    [ 1, -1, -1, -1,  1]
]))
```

Após essa etapa, o modelo Adaline foi implementado com os métodos de função de ativação, predição e treino - este último onde a operação dos mínimos métodos quadrados foi realizada - como demonstrado no Listing 2.

Listing 2: Implementação do modelo Adaline das variáveis

```
    # Adaline class
class Adaline:
    def __init__(self, input_size, learning_rate=0.01, epochs=100):
        self.weights = np.zeros(input_size)
        self.bias = 0.0
        self.learning_rate = learning_rate
        self.epochs = epochs

    def activation_function(self, x):
        return x  # Linear function activation

    def predict(self, X):
        linear_output = np.dot(X, self.weights) + self.bias
        return np.where(linear_output >= 0.0, 1, -1)

    def train(self, X, y):
        for epoch in range(self.epochs):
            outputs = self.activation_function(np.dot(X, self.weights) + self.bi
            errors = y - outputs
            self.weights += self.learning_rate * np.dot(X.T, errors)
            self.bias += self.learning_rate * errors.sum()

            # Calculating Mean Squared error
            mse = (errors**2).mean()
            print(f"Epoch {epoch+1}/{self.epochs} - MSE: {mse:.4f}")

# Initializing model
adaline = Adaline(input_size=25, learning_rate=0.001, epochs=20)
```

Utilizando-se da base de treino, o modelo foi criado e treinado com 20 iterações resultando em um erro quadrático médio de 0.3973.

Por fim, foram criadas para base de teste doze variáveis, seis para Y e seis para Y invertido, com ruido inserido manualmente. O modelo implementado foi

aplicado à base de teste, resultando em uma precisão de 91.76%. A saída para cada amostra, assim como a precisão podem ser visualizadas abaixo no Listing 3.

Listing 3: Resultados

```
Sample 1: Prediction = Y, Real = Y
Sample 2: Prediction = Y, Real = Y
Sample 3: Prediction = Y, Real = Y
Sample 4: Prediction = Y, Real = Y
Sample 5: Prediction = Y, Real = Y
Sample 6: Prediction = Y, Real = Y
Sample 7: Prediction = Inverted Y, Real = Inverted Y
Sample 8: Prediction = Inverted Y, Real = Inverted Y
Sample 9: Prediction = Inverted Y, Real = Inverted Y
Sample 10: Prediction = Inverted Y, Real = Inverted Y
Sample 11: Prediction = Inverted Y, Real = Inverted Y
Sample 12: Prediction = Y, Real = Inverted Y
Total score prediction: 91.67%
```

A implementação inteira do código, assim como a base de teste criada, pode ser visualizada no repositório criado no *GitHub*[3], onde também contêm instruções para a execução do código.

[3]Repositório do Código Disponível em: https://github.com/MatheusUema/neural-networks-study. Acesso em: 27 de ago. de 2024.