Instituto Politécnico Nacional

Neurofuzzy Systems

PRACTICE NN 1. PERCEPTRON VS ADALINE

López Favila Carlos Josafath
Delivery Date: - May 2023

I. Introduction

In the realm of artificial intelligence and machine learning, neural networks have emerged as a powerful paradigm for solving complex problems. Inspired by the structure and functionality of the human brain, neural networks simulate the behavior of interconnected neurons to process and analyze vast amounts of data. With their ability to learn from experience, adapt, and generalize, neural networks have revolutionized various domains, including image recognition, natural language processing, and pattern classification.
The Perceptron, introduced by Frank Rosenblatt in 1957, can be considered the precursor to modern neural networks. It consists of a single layer of artificial neurons, or perceptrons, interconnected through weighted connections. This simple yet powerful model paved the way for the development of more complex neural network architectures. However, the Perceptron has limitations in handling linearly non-separable data and updating its weights in real-time, leading to the need for further advancements.
Enter Adaline, which stands for Adaptive Linear Neuron. Developed by Bernard Widrow and Tedd Hoff in 1960, Adaline builds upon the foundation laid by the Perceptron. It addresses some of the Perceptron's shortcomings by introducing a more flexible learning rule and the concept of continuous activation. Adaline's breakthrough lies in its ability to handle linearly non-separable data through gradient descent optimization, allowing for fine-tuning of weights and improved convergence.
By understanding the differences between Perceptron and Adaline, we gain insights into the progression of neural networks and the evolving strategies employed to address various challenges. This exploration will equip us with a solid foundation to tackle real-world problems using neural network algorithms and open doors to further advancements in artificial intelligence and machine learning.

II. Objective

Objective of the Practice:
The objective of this practice is to gain a practical understanding of neural networks by focusing on the Perceptron and Adaline models. Through hands-on exploration and experimentation, the practice aims to achieve the following objectives:
1. Implement Perceptron and Adaline algorithms: Develop a working knowledge of how to implement the Perceptron and Adaline algorithms using appropriate programming tools or frameworks. This will involve understanding the mathematical foundations of these algorithms and translating them into code.
2. Explore architecture and learning rules: Investigate the architectural differences between Perceptron and Adaline, including the number of layers, activation functions, and learning rules. Analyze how these differences affect the models' capabilities, convergence, and generalization.
3. Compare performance on linearly separable and non-separable data: Evaluate the performance of Perceptron and Adaline models on linearly separable and non-separable datasets. Analyze and compare their abilities to correctly classify and converge on the correct decision boundaries for different types of data.
4. Examine convergence and limitations: Study the convergence properties of Perceptron and Adaline models, including their limitations in handling complex or overlapping classes. Identify scenarios where one model outperforms the other and understand the factors contributing to these differences.
5. Gain insights into practical applications: Discuss the real-world applications of Perceptron and

Adaline models, considering their strengths and weaknesses. Explore scenarios where one model may be more suitable than the other based on the characteristics of the problem and available data.

III. Development

It is asked to implement:

A. Perceptron

Implement a percepron neural netwokr to classify the toys (bears and rabbits). Train the network for different number of epochs:

- 20
- 50
- 100

B. Adaline

Implement an Adaline neural network to classify the toys (bear and rabbits). Train the network for different number of epochs:

1. 20

- alpha=1/(4*lambdamax)
- alpha=1/(8*lambdamax)
- alpha=1/(16*lambdamax)

2. 50

- alpha=1/(4*lambdamax)
- alpha=1/(8*lambdamax)
- alpha=1/(16*lambdamax)

3. 100

- alpha=1/(4*lambdamax)
- alpha=1/(8*lambdamax)
- alpha=1/(16*lambdamax)

Then make the comparision of Perceptron vs Adaline. Start with random values of W and B, the same for Perceptron and Adaline. Train both networks simultaneously for different number of epochs and different values oof alpha in Adaline. In one figure subplot for each of the number of epochs (3 graphs):

- Solution of the 2 networks(patterns, limit lines (overlaped in different colour))

- Error perceptron (1 subplot), 3 errors for adaline(1 subplot, in different colours). Add labels and titles to your graphs.

  To test the networks:

  - Select the best option for alpha and number of epochs
  - Ask the user for a input(p)
  - Calculate the output for perceptron and adaline
  - Display a figure of bear or rabbit

IV. Results

The first results we got are the subplots of the solutions of both Networks.

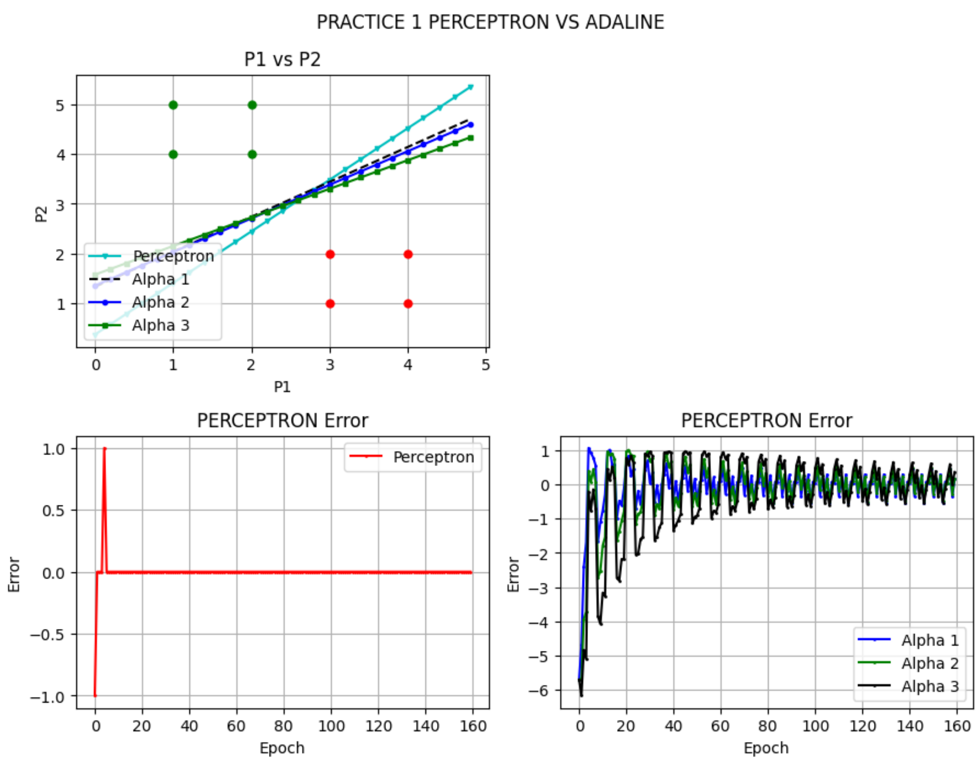

Figure 1: 20 Epochs

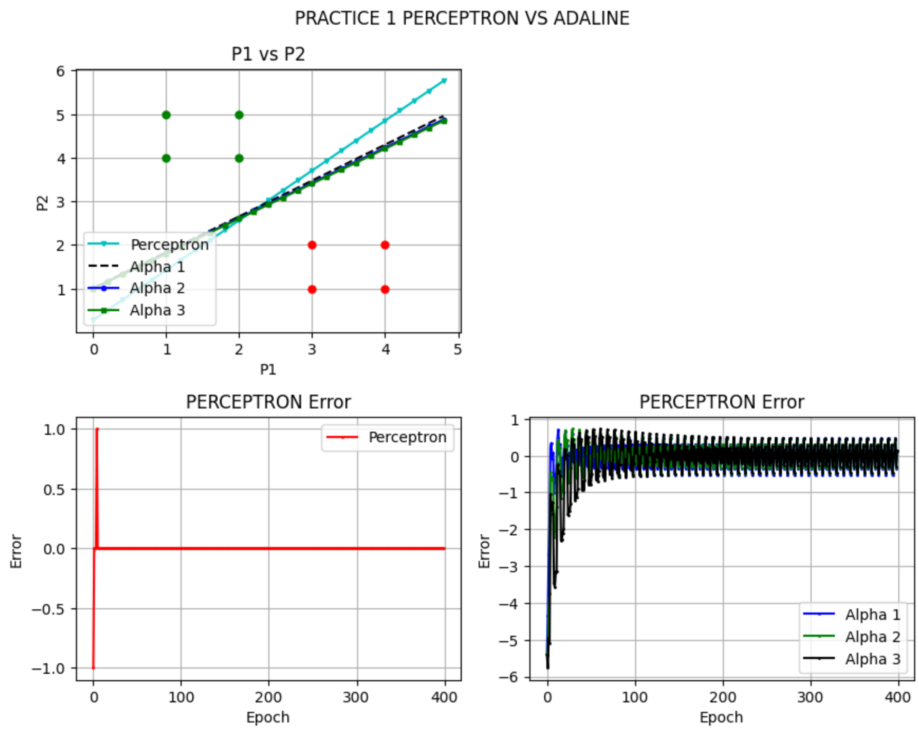

Figure 2: 50 Epochs

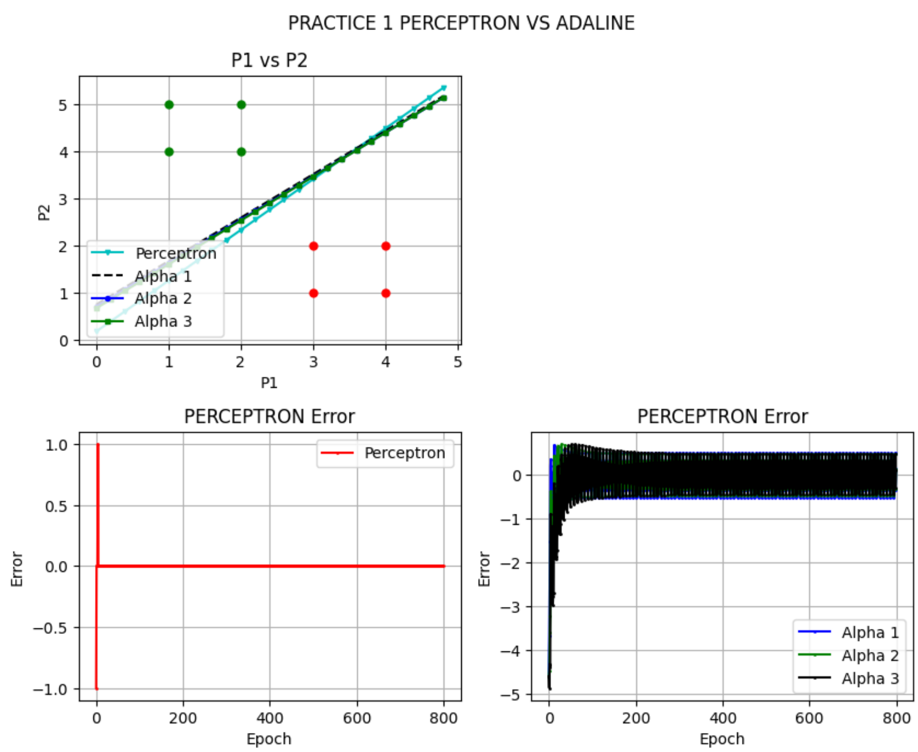

Figure 3: 100 Epoch

As it can be seen, the difference between Perceptron and Adaline is that in perceptron, once it founds a solution, it does not try to find another better solution, that's why the error is almost always 0, in contrast with adaline's error. that it begins in -5 and it tends to turn around 0, but it is always chaging trying to find a better solution.
In this cases, I considered that the best solution was with Adaline and with the second alpha and with 100 epochs. So in the testing we got the next results:

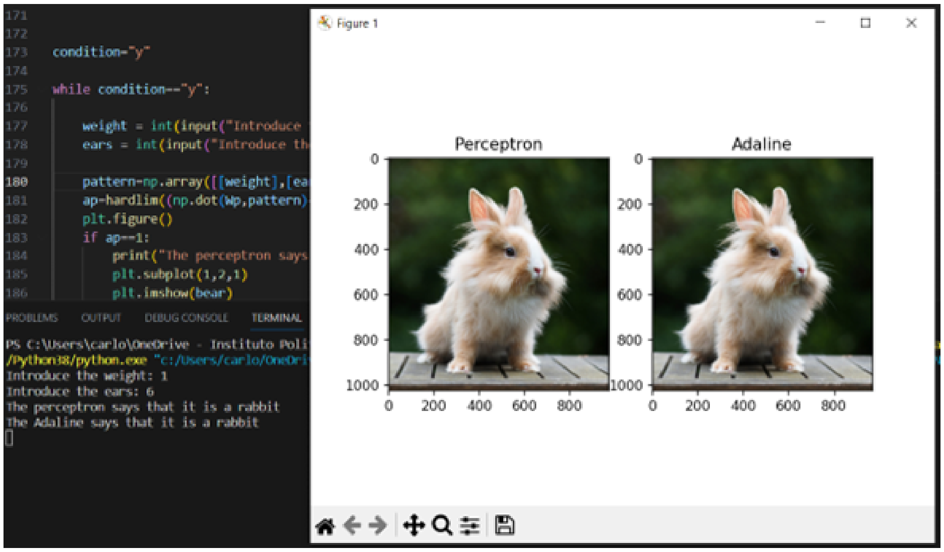

Figure 4: Case of Rabbit

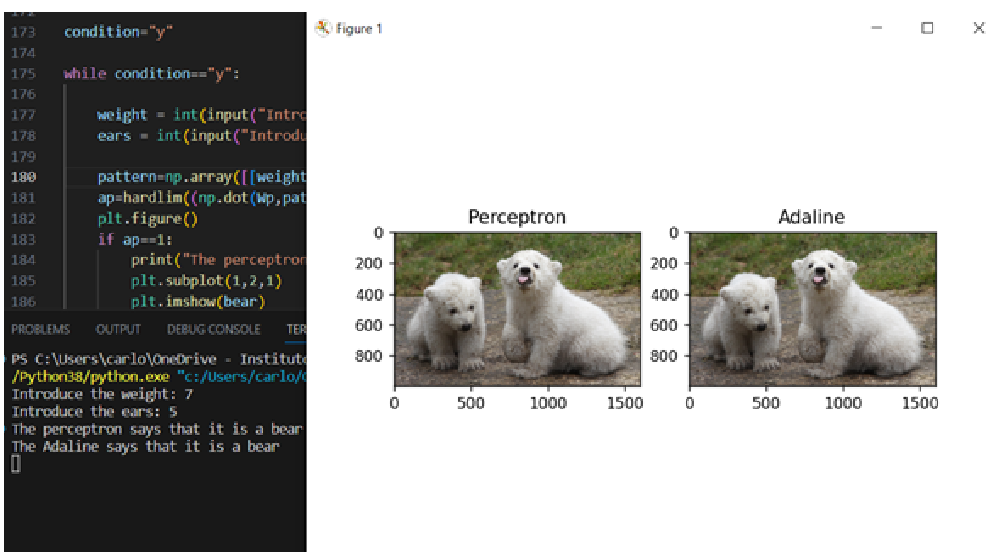

Figure 5: Case of Bear

V. Conclusions

In conclusion, the practice aimed to understand the differences between the perceptron and Adaline algorithms. Both algorithms are used for binary classification tasks, but they differ in terms of their

learning rule and decision boundaries.
The perceptron algorithm uses a simple step function as its activation function and employs the perceptron learning rule, which updates the weights based on misclassified samples. It has a linear decision boundary and can only solve linearly separable problems. However, it converges to a solution if the data is linearly separable.
On the other hand, Adaline (Adaptive Linear Neuron) uses a linear activation function and employs the Widrow-Hoff learning rule, also known as the delta rule or least mean squares (LMS) algorithm. Adaline aims to minimize the mean squared error between its predicted outputs and the desired outputs. Unlike the perceptron, Adaline can handle non-linearly separable problems because it uses continuous outputs and computes a weighted sum of inputs rather than a binary output.
Overall, the practice helped to highlight the key differences between the perceptron and Adaline algorithms, such as their learning rules and decision boundaries, and provided a hands-on understanding of their behavior on different types of data.—

VI. Apenddix

A. Code

1. Code

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

def hardlim(x):
    if (x<0).all():
        return 0
    else:
        return 1

def purelin(x):
    return x


# patterns
P=np.array([[1,1,2,2,3,3,4,4], [4,5,4,5,1,2,1,2]]) #matrix of patterns
Tp=np.array([0,0,0,0,1,1,1,1])
Ta=np.array([-1,-1,-1,-1,1,1,1,1])

[R,N]=P.shape
S=1 # number of neurons
epoch=100 # number of epochs

#WEIGHTS AND BIAS FOR PERCEPTRON
Wp=np.random.rand(S,R) # matrix of weights
bp=np.random.rand(S,1) # matrix of bias

#WEIGHTS AND BIAS FOR ADALINE
Wa1=Wp
ba1=bp
Wa2=Wp
ba2=bp
Wa3=Wp
ba3=bp
```

```python

R=(1/N)*np.dot(P,P.T)
eigen=np.linalg.eigvals(R)

#Alpha
alpha1=1/(4*max(eigen))
alpha2=1/(8*max(eigen))
alpha3=1/(16*max(eigen))

#errors list
errorp=[]
errora1=[]
errora2=[]
errora3=[]

it=0
ue=[]
#Training
for i in range(epoch):
    for j in range(N):
        ue.append(it)
        it=it+1

        x=P[:,j]

        #Percetron Training
        ap=hardlim((np.dot(Wp,x)+bp))
        e_p=Tp[j]-ap
        errorp.append(e_p)
        Wp=Wp+e_p*x
        bp=bp+e_p

        #Adeline Training 1
        aa1=purelin((np.dot(Wa1,x)+ba1))
        e_a1=Ta[j]-aa1
        errora1.append(e_a1[0][0])
        Wa1=Wa1+alpha1*e_a1*x
        ba1=ba1+alpha1*e_a1

        #Adeline Training 2
        aa2=purelin((np.dot(Wa2,x)+ba2))
        e_a2=Ta[j]-aa2
        errora2.append(e_a2[0][0])
        Wa2=Wa2+alpha2*e_a2*x
        ba2=ba2+alpha2*e_a2

        #Adeline Training 3
        aa3=purelin((np.dot(Wa3,x)+ba3))
        e_a3=Ta[j]-aa3
        errora3.append(e_a3[0][0])
        Wa3=Wa3+alpha3*e_a3*x
        ba3=ba3+alpha3*e_a3

X = np.arange(0,5,0.2)
Y = np.arange(0,5,0.2)

#--------------------------------------------------------------PLOT PATTERNS
a=P.tolist()

```

```
fig=plt.figure(figsize=(10,10),tight_layout=True)
fig.suptitle("PRACTICE 1 PERCEPTRON VS ADALINE")
plt.subplot(2,2,1)
for i in range(len(a[0])):
    if i <=3:
        plt.plot(a[0][i],a[1][i],'go',markersize=5)
    else:
        plt.plot(a[0][i],a[1][i],'ro',markersize=5)

#---------------------------------------------------------------LIMIT LINE ALPHA1
Wpp=Wp[0].tolist()
bpp=bp[0].tolist()

limlineP=[]
for x in X:
    limlineP.append(-bpp[0]/Wpp[1]-Wpp[0]/Wpp[1]*x)
plt.plot(X,limlineP,'cv-', markersize=3,label='Perceptron')

#---------------------------------------------------------------LIMIT LINE ALPHA1
Wa1p=Wa1[0].tolist()
ba1p=ba1[0].tolist()

limlineA1=[]
for x in X:
    limlineA1.append(-ba1p[0]/Wa1p[1]-Wa1p[0]/Wa1p[1]*x)
plt.plot(X,limlineA1,'k--', markersize=3,label='Alpha 1')

#---------------------------------------------------------------LIMIT LINE ALPHA2
Wa2p=Wa2[0].tolist()
ba2p=ba2[0].tolist()

limlineA2=[]
for x in X:
    limlineA2.append(-ba2p[0]/Wa2p[1]-Wa2p[0]/Wa2p[1]*x)
plt.plot(X,limlineA2,'b-o', markersize=3,label='Alpha 2')

#---------------------------------------------------------------LIMIT LINE ALPHA3
Wa3p=Wa3[0].tolist()
ba3p=ba3[0].tolist()

limlineA3=[]
for x in X:
    limlineA3.append(-ba3p[0]/Wa3p[1]-Wa3p[0]/Wa3p[1]*x)
plt.plot(X,limlineA3,'g-s', markersize=3,label='Alpha 3')
plt.xlabel('P1')
plt.ylabel('P2')
plt.title('P1 vs P2')
plt.legend(loc="lower left")
plt.grid(True)

#---------------------------------------------------------------PLOT PERCETRON ERROR

plt.subplot(2,2,3)
plt.plot(ue,errorp,'r-o',markersize=1,label='Perceptron')
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.title('PERCEPTRON Error')
plt.legend()
plt.grid(True)
```

```
#---------------------------------------------------------------PLOT ADALINE ERROR

plt.subplot(2,2,4)
plt.plot(ue,errora1,'b-o',markersize=1,label='Alpha 1')
plt.plot(ue,errora2,'g-o',markersize=1,label='Alpha 2')
plt.plot(ue,errora3,'k-o',markersize=1,label='Alpha 3')
plt.xlabel('Epoch')
plt.ylabel('Error')
plt.title('PERCEPTRON Error')
plt.grid(True)

#plt.show()


#---------------------------------------------------------------TESTING
bear = mpimg.imread('bear.jpg')
rabbit = mpimg.imread('rabbit.jpg')

print(Wp)
print(Wa2)

condition="y"

while condition=="y":

    weight = int(input("Introduce the weight: "))
    ears = int(input("Introduce the ears: "))

    pattern=np.array([[weight],[ears]])
    print(pattern)
    ap=hardlim((np.dot(Wp,pattern)+bp))
    plt.figure()
    if ap==1:
        print("The perceptron says that it is a bear")
        plt.subplot(1,2,1)
        plt.imshow(bear)
        plt.title("Perceptron")
    else:
        print("The perceptron says that it is a rabbit")
        plt.subplot(1,2,1)
        plt.imshow(rabbit)
        plt.title("Perceptron")

    aa=purelin((np.dot(Wa2,pattern)+ba2))
    if aa>0:
        print("The Adaline says that it is a bear")
        plt.subplot(1,2,2)
        plt.imshow(bear)
        plt.title("Adaline")
    else:
        print("The Adaline says that it is a rabbit")
        plt.subplot(1,2,2)
        plt.imshow(rabbit)

    plt.show()
    condition=input("Do you want to continue? (y/n): ")

print("BYE!")
```