Instituto Politécnico Nacional

Neurofuzzy Systems

PRACTICE NN 1. PERCEPTRON VS ADALINE

López Favila Carlos Josafath
Delivery Date: - June 2023

I. Introduction

Neural networks are a type of computational model inspired by the structure and functioning of biological neural networks, such as the human brain. They are composed of interconnected artificial neurons, or nodes, organized in layers. Each neuron takes input signals, applies a transformation to them, and produces an output signal that is passed to the next layer. Neural networks are designed to learn from data and make predictions or decisions based on the patterns and relationships they discover.
Multilayer neural networks, also known as deep neural networks or deep learning models, are neural networks with multiple layers of neurons between the input and output layers. These layers, called hidden layers, allow the network to learn more complex representations of the data. Each neuron in a hidden layer receives inputs from the previous layer and applies a nonlinear activation function to produce an output. The outputs of the neurons in one layer serve as inputs to the neurons in the subsequent layer until the final output is generated.
The advantage of using multilayer neural networks is their ability to learn hierarchical representations of data, capturing intricate patterns and relationships. With deeper architectures, these networks can extract and transform features from raw data, enabling them to solve complex tasks such as image recognition, natural language processing, and speech recognition. Multilayer neural networks employ various training algorithms, such as backpropagation, to adjust the weights and biases of the neurons during the learning process, minimizing the difference between the predicted and actual outputs.
The recent advancements in computing power, availability of large-scale datasets, and algorithmic improvements have led to significant breakthroughs in multilayer neural networks. They have revolutionized fields like computer vision, natural language processing, and many others, achieving state-of-the-art performance in various applications. Their versatility, combined with their ability to learn from large amounts of data, makes them a powerful tool for solving complex problems and pushing the boundaries of artificial intelligence.

II. Objective

Objective of the Practice:
The objective of this practice is to gain a practical understanding of multilayer neural networks
Network Architecture Design: Design the architecture of a multilayer neural network suitable for image generation. Determine the number of layers, the number of neurons in each layer, and the appropriate activation functions to be used.

III. Development

In the development we solve the problem we have. To create a Happy face with multilayer neural networks. In this case I did all the calculations on the code directly. So all the line equations, weight and bias calculations for each neuron and layer could be verified in the anexed code in the Appendix Section.

IV. Results

As a result we got the following hapy face. It is built with 2 triangles as the eyes. and a mouth built with 1 rectangle and 2 triangles.

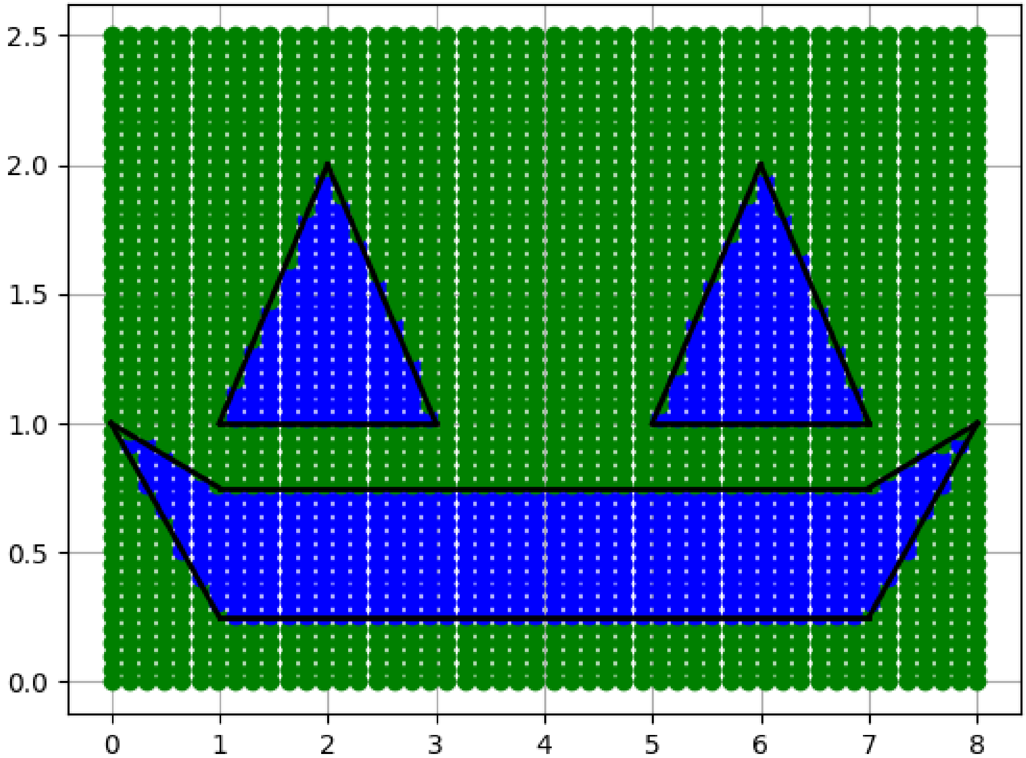

Figure 1: Happy Face Result.

V. Apenddix

A. Code

1. Code

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

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

#------------------------------------------------------------LEFT EYE--------------------------------------------------
#proposing 3 points as the triangle vertices
#P1=[2,2]
#P2=[1,1]
#P3=[3,1]

#so, for this we have 3 lines (neurons):
```

```
# Yl1 - 2= (2-1/2-1)*(X-2) -> Yl1 = X
# Yl2 - 2= (2-1/2-3)*(x-2) -> Yl2 = -X+4
# YL3 = 1

#So, the weight line equations are:
# Yw11 = -X
# Yw12 = X
# Yw13 = inf

# so, we want the line of the weight 1 points at the middle of the three lines
# proposing x = 1, with Yw11=-x we get Yw11 = -1

# and we also want the line of the weight 2 points at the middle of the thre lines
# proposing x = -1, with Yw12=x we get Yw12 = -1

# and we also want the line of the weight 3 points at the middle of the thre lines
# in this case, the weight 3 is a vertical line, so we can propose x=0 and in Yw13=1

w11=[1,-1]
w12=[-1,-1]
w13=[0,1]

# calculating bias for lef eye
# a point on line 1 is [0,0]
b11=-np.dot(w11,[0,0])

# a point on line 2 is [0,4]
b12=-np.dot(w12,[0,4])

# a point on line 3 is [0,1]
b13=-np.dot(w13,[0,1])

#-----------------------------------------------------------RIGHT EYE--------------------------------------------------
#proposing 3 points as the triangle vertices
#P4=[6,2]
#P5=[5,1]
#P6=[7,1]

#so, for this we have 3 lines (neurons):
# Yl4 - 2= (2-1/6-5)*(X-6) -> Yl4 = X-4
# Yl5 - 2= (2-1/6-7)*(x-6) -> Yl5 = -X+8
# Yl6 = 1 = Yl3

#So, the weight line equations are:
# Yw14 = -X
# Yw15 = X
# Yw16 = inf

# so, we want the line of the weight 4 points at the middle of the three lines
# proposing x = 1, with Yw14=-x we get Yw14 = -1

# and we also want the line of the weight 5 points at the middle of the thre lines
# proposing x = -1, with Yw15=x we get Yw15 = -1

# and we also want the line of the weight 6 points at the middle of the thre lines
# in this case, the weight 3 is a vertical line, so we can propose x=0 and in Yw16=1

# we can see that we have the same weights as the left eye, so we can use the same weights

```

```
w14,w15,w16=w11,w12,w13
# calculating bias for right eye

# a point on line 1 is [0,-4]
b14=-np.dot(w14,[0,-4])

# a point on line 2 is [0,8]
b15=-np.dot(w15,[0,8])

# a point on line 3 is [0,1]
b16=-np.dot(w16,[0,1])

#------------------------------------------------------------MOUTH----------------------------------------------------------------

#so, for this we have 4 lines (neurons):
# Xl7 = 1 (vertical line)
# Xl8 = 7 (vertical line)
# Yl9 = 0.25
# Yl10 = 0.75
# Yl11 = -(1/4)*X + 1 -> wl11=1/(1/4)*X -> wl11=[-1,-4]
# Yl12 = -(3/4)*X + 1 -> wl12=1/(3/4)*X -> wl12=[1,4/3]
# Xl13 = 1 (vertical line)
# Yl14 = (1/4)*X + 1 - > wl14=-1/(1/4)*X -> wl14=[1,-4]
# Yl15 = (3/4)*X + 5 - > wl15=-1/(3/4)*X -> wl15=[-1,4/3]
# Yl16 = 7 (vertical line)


#So, the weights are:
w17=[1,0]
w18=[-1,0]
w19=[0,1]
w110=[0,-1]
w111=[-1,-4]
w112=[1,4/3]
w113=w18
w114=[1,-4]
w115=[-1,4/3]
w116=w17


#CALCULATING THE BIAS's
#for point on the fourth limit line (neuron 7) P7[1,0.75]
b17=-np.dot(w17,[1,0.75])

#for point on the fifth limit line (neuron 8) P8[7,0.25]
b18=-np.dot(w18,[7,0.25])

#for point on the sixth limit line (neuron 9) P9[3.5,0.25]
b19=-np.dot(w19,[3.5,0.25])

#for point on the seventh limit line (neuron 10) P10[3.5,0.75]
b110=-np.dot(w110,[3.5,0.75])

#for point on the eighth limit line (neuron 11) P11[0,(7/8)]
b111=-np.dot(w111,[0,1])

#for point on the ninth limit line (neuron 12) P12[0,(5/8)]
b112=-np.dot(w112,[0,1])

```

```
#for point on the tenth limit line (neuron 13) P13[1,(0.5)]
b113=-np.dot(w113,[1,0.5])

#for point on the eleventh limit line (neuron 14) P14[7,(0.75)]
b114=-np.dot(w114,[7,0.75])

#for point on the twelfth limit line (neuron 15) P15[7,(0.25)]
b115=-np.dot(w115,[7,0.25])

#for point on the thirteenth limit line (neuron 16) P16[7,(0.5)]
b116=-np.dot(w116,[7,0.5])



W1=[w11,w12,w13,w14,w15,w17,w18,w19,w110,w111,w112,w113,w114,w115,w116]
B1=[[b11],[b12],[b13],[b14],[b15],[b17],[b18],[b19],[b110],[b111],[b112],[b113],[b114],[b115],[b116]]


#------------------------------------------------------------2nd layer AND-----------------------------------------------
#W2=[[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0], # this line is the sum of the weights of the left eye
#    [0,0,1,1,1,0,0,0,0,0,0,0,0,0,0], # this line is the sum of the weights of the right eye
#    [0,0,0,0,0,1,1,1,1,0,0,0,0,0,0], # this line is the sum of the weights of the mouth's rectangle
#    [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0], # this line is the sum of the weights of the mouth's left triangle
#    [0,0,0,0,0,0,0,0,0,0,0,0,1,1,1]] # this line is the sum of the weights of the mouth's right triangle

#W*P+b>=0 MINIMUM CONDITION TO GET '1'
#W2=[[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0], *[[a1],[a2],[a3],[a4],[a5],[a7],[a8],[a9],[a10],[a11],[a12],[a13],[a14],[a15]
#    [0,0,1,1,1,0,0,0,0,0,0,0,0,0,0],
#    [0,0,0,0,0,1,1,1,1,0,0,0,0,0,0],
#    [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0],
#    [0,0,0,0,0,0,0,0,0,0,0,0,1,1,1]]

#W*P+b<0 MAXIMUM CONDITION TO GET '0'
#W2=[[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0], *[[a1],[a2],[a3],[a4],[a5],[a7],[a8],[a9],[a10],[a11],[a12],[a13],[a14],[a15]
#    [0,0,1,1,1,0,0,0,0,0,0,0,0,0,0],
#    [0,0,0,0,0,1,1,1,1,0,0,0,0,0,0],
#    [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0],
#    [0,0,0,0,0,0,0,0,0,0,0,0,1,1,1]]

# for the right eye
#[1,1,1]*[[a1],[a2],[a3]]+b>=0 # to get '1'
#[1,1,1]*[[1],[1],[1]]+b>=0
# 3+b>=0 -> b>=-3
#[1,1,1]*[[a1],[a2],[a3]]+b<0 # to get '0'
#[1,1,1]*[[1],[1],[0]]+b<0
# 2+b<0 -> b<-2
# -3<=b<-2 -> b=-2.5

# for the left eye
#[1,1,1]*[[a3],[a4],[a5]]+b>=0 # to get '1'
#[1,1,1]*[[1],[1],[1]]+b>=0
# 3+b>=0 -> b>=-3
#[1,1,1]*[[a3],[a4],[a5]]+b<0 # to get '0'
#[1,1,1]*[[1],[1],[0]]+b<0
# 2+b<0 -> b<-2
# -3<=b<-2 -> b=-2.5

#for the mouth's rectangle
#[1,1,1,1]*[[a7],[a8],[a19],[a10]]+b>=0 # to get '1'
```

```
#[1,1,1,1]*[[1],[1],[1],[1]]+b>=0
# 4+b>=0 -> b>=-4

#[1,1,1,1]*[[a7],[a8],[a19],[a10]]+b<0 # to get '0'
#[1,1,1,1]*[[1],[1],[1],[0]]+b<0
# 3+b<0 -> b<-3
# -4<=b<-3 -> b=3.5

#for the mouth's triangles
#[1,1,1]*[[a11],[a12],[a13]]+b>=0 # to get '1'
#[1,1,1]*[[1],[1],[1]]+b>=0
# 3+b>=0 -> b>=-3

#[1,1,1]*[[a11],[a12],[a13]]+b<0 # to get '0'
#[1,1,1]*[[1],[1],[0]]+b<0
# 2+b<0 -> b<-2
# -3<=b<-2 -> b=-2.5


W2=[[1,1,1,0,0,0,0,0,0,0,0,0,0,0,0],
    [0,0,1,1,1,0,0,0,0,0,0,0,0,0,0],
    [0,0,0,0,0,1,1,1,1,0,0,0,0,0,0],
    [0,0,0,0,0,0,0,0,0,1,1,1,0,0,0],
    [0,0,0,0,0,0,0,0,0,0,0,0,1,1,1]]


B2=[[-2.5],[-2.5],[-3.5],[-2.5],[-2.5]]

#------------------------------------------------------------3rd layer OR--------------------------------------------

W3=[1,1,1,1,1] # all inputs have the same weight (same importance)

#W*P+b>=0 MINIMUM CONDITION TO GET '1'
#[1,1,1,1]*[[1],[0],[0],[0]]+b>=0 or [1,1,1]*[[0],[1],[0],[0]]+b>=0 or [1,1,1]*[[0],[0],[1],[0]]+b>=0 or [1,1,1]*[[
# 1+b>=0 -> b>=-1

#W*P+b<0 MAXIMUM CONDITION TO GET '0'
#[1,1,1,1]*[[0],[0],[0],[0]]+b<0
# 0+b<0 -> b<0

#-1<=b<0 -> b=-0.5

B3=[-0.5]

#------------------------------------------------------------PLOTTING------------------------------------------------
xl1=np.arange(1,2.1,0.1)
yl1=xl1

xl2=np.arange(2,3.1,0.1)
yl2=-xl2+4

xl3=np.arange(1,3.1,0.1)
yl3=[]
for i in range(len(xl3)):
    yl3.append(1)

xl4=np.arange(5,6.1,0.1)
yl4=xl4-4
```

```
xl5=np.arange(6,7.1,0.1)
yl5=-xl5+8

xl6=np.arange(5,7.1,0.1)
yl6=[]

for i in range(len(xl6)):
    yl6.append(1)

xl9=np.arange(1,7.1,0.1)
yl9=[]
for i in range(len(xl9)):
    yl9.append(0.75)

xl10=np.arange(1,7.1,0.1)
yl10=[]
for i in range(len(xl10)):
    yl10.append(0.25)

xl11=np.arange(0,1.1,0.1)
yl11=-(1/4)*xl11+1

xl12=np.arange(0,1.1,0.1)
yl12=-(3/4)*xl12+1

xl13=np.arange(7,8.1,0.1)
yl13=(1/4)*xl13-1

xl14=np.arange(7,8.1,0.1)
yl14=(3/4)*xl14-5

plt.figure(1)

xp=np.linspace(0,8)
yp=np.linspace(0,2.5)

for x in xp:
    for y in yp:
        a1=np.dot(W1,[[x],[y]])+B1
        for i in range(len(a1)):
            a1[i]=hardlim(a1[i])
        a2=np.dot(W2,a1)+B2
        for i in range(len(a2)):
            a2[i]=hardlim(a2[i])
        a3=hardlim(np.dot(W3,a2)+B3)

        if a3==1:
            plt.plot(x,y,'bo')
        else:
            plt.plot(x,y,'go')


plt.plot(xl1,yl1,'k',xl2,yl2,'k',xl3,yl3,'k',xl4,yl4,'k',xl5,yl5,'k',xl6,yl6,'k',
         xl9,yl9,'k',xl10,yl10,'k',xl11,yl11,'k',xl12,yl12,'k',xl13,yl13,'k',xl14,yl14,'k',linewidth=2.0)
plt.grid(True)

plt.show()
```