ECE-CS 559 Neural Networks

Homework 2

Virginia Tasso

Professor: Mesrob Ohannessian

September 10, 2024

1 Exercise 1

1.1 Point a)

Build the neural network that implements the following logic:

$$(x_1 \wedge x_2 \wedge \neg x_3) \vee (\neg x_2 \wedge x_3)$$

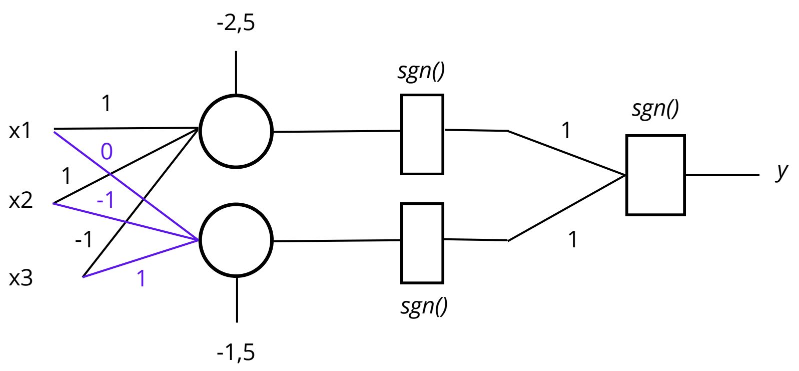

Figure 1: Neural network architecture designed to implement the logic $x_1 \wedge x_2 \wedge \neg x_3$. It is a 2-2-1-1 feed-forward neural network: 2 layers, with 2 neurons in the first layer, 1 in the second, and 1 output.

The neural network has the following parameters:

$$W = \begin{pmatrix} 1 & 1 & -1 \\ 0 & -1 & 1 \end{pmatrix}$$

The rows of the matrix W represent the neurons, while the columns represent the inputs.

$$b = \begin{pmatrix} -2.5 \\ -1.5 \end{pmatrix}$$

The rows represent the two neurons as for W.

$$u = \begin{pmatrix} 1 & 1 \end{pmatrix}$$

In the second layer, the input is the output of the first layer. Since there is only one neuron in the second layer, there is only one row but two columns representative of two inputs.

$$c = 0.5$$

The final bias is a scalar.

1.2 Point b)

Write down the input-to-output analytic equation that your network represents.

The network is composed of two layers. The weights $w_{11}, w_{12}, ..., w_{23}$ are associated with the first layer, while u_1, u_2 are used in the second layer. ϕ is the activation function, which is, in this case, the *sign* function.

For the first layer, the outputs are computed as:

$$z_1 = \phi(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1)$$

This is output of neuron 1

$$z_2 = \phi(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2)$$

This is output of neuron 2

The first subscript is relative to the neuron, while the second is relative to the input.

The final output is computed as:

$$y = \phi(u_1 z_1 + u_2 z_2 + c)$$

Where z_1 and z_2 are the outputs from the first layer, and c is the bias term for the second layer.

By expanding the expression:

$$y = \phi(u_1 \cdot \phi(w_{11}x_1 + w_{12}x_2 + w_{13}x_3 + b_1) + u_2 \cdot \phi(w_{21}x_1 + w_{22}x_2 + w_{23}x_3 + b_2) + c)$$

1.3 Point c)

```python
# --------- import useful libraries ---------

import itertools
import numpy as np
import pandas as pd

# --------- neural network parameters -------

w = np.array([[1, 1, -1], [0, -1, 1]]) # weights matrix
b = np.array([-2.5, -1.5])
u = np.array([1, 1])
c = 0.5


# --------- define useful functions ---------
# function to implement logic gate

def logic_gate(inputs):
    x1, x2, x3 = inputs
    # need to manage the fact that inputs can be -1. The logic gate
   works with 1 and 0 --> conversion of the values
    x1, x2, x3 = (0 if x == -1 else x for x in inputs)
    return bool(x1 and x2 and not x3) or bool(not x2 and x3)

```

```python

# function to implement the analitic expression

def f_x(inputs):
    x1, x2, x3 = inputs
    inputs = np.array([x1, x2, x3])
    first = np.add(np.matmul(w, inputs),b)
    first = np.sign(first)
    second = np.add(np.matmul(u, first), c)
    res = int(np.sign(second))

    if res == 1: return True
    else: return False




perms = list(itertools.product([1,-1], repeat = 3))
perms = np.asarray(perms)


# --------- build the table resulting from the analitic expression and
    from the logic gate---------

print ("\t x1 | x2 | x3 | output\n---------------------------------")

data_1 = [] # data with the outputs of the analytic expression
data_2 = [] # data with the outputs of the logic gate
for perm in perms:
    x1, x2,x3 = perm
    y1 = f_x(np.array([x1, x2, x3]))  # analitic expression
    y2 = logic_gate(np.array([x1, x2, x3])) # logic gate
    print(f"\t{x1} | {x2} | {x3} | {y1} ")
    data_1.append([x1, x2, x3, y1])
    data_2.append([x1, x2, x3, y2])

df1 = pd.DataFrame(data_1, columns=['x1', 'x2', 'x3', 'Output'])
df2 = pd.DataFrame(data_2, columns=['x1', 'x2', 'x3', 'Output'])

# check if the two tables match

print(df1['Output'].equals(df2['Output']))
```

The resulting logic table is shown below:

x_1	x_2	x_3	**Output**
1	1	1	False
1	1	-1	True
1	-1	1	True
1	-1	-1	False
-1	1	1	False
-1	1	-1	False
-1	-1	1	True
-1	-1	-1	False

The output of the analytic expression:

$$y = \phi(1 \cdot \phi(x_1 + x_2 - x_3 - 2.5) + 1 \cdot \phi(0 \cdot x_1 - x_2 + x_3 - 1.5) + 0.5)$$

x_1	x_2	x_3	**Output**
1	1	1	$\phi(1 \cdot \phi(1 \cdot 1 + 1 \cdot 1 - 1 \cdot 1 - 2.5) + 1 \cdot \phi(0 \cdot 1 - 1 \cdot 1 + 1 \cdot 1 - 1.5) + 0.5) = \text{False}$
1	1	-1	$\phi(1 \cdot \phi(1 \cdot 1 + 1 \cdot 1 - 1 \cdot (-1) - 2.5) + 1 \cdot \phi(0 \cdot 1 - 1 \cdot 1 + 1 \cdot (-1) - 1.5) + 0.5) = \text{True}$
1	-1	1	$\phi(1 \cdot \phi(1 \cdot 1 + 1 \cdot (-1) - 1 \cdot 1 - 2.5) + 1 \cdot \phi(0 \cdot 1 - 1 \cdot (-1) + 1 \cdot 1 - 1.5) + 0.5) = \text{True}$
1	-1	-1	$\phi(1 \cdot \phi(1 \cdot 1 + 1 \cdot (-1) - 1 \cdot (-1) - 2.5) + 1 \cdot \phi(0 \cdot 1 - 1 \cdot (-1) + 1 \cdot (-1) - 1.5) + 0.5) = \text{False}$
-1	1	1	$\phi(1 \cdot \phi(1 \cdot (-1) + 1 \cdot 1 - 1 \cdot 1 - 2.5) + 1 \cdot \phi(0 \cdot (-1) - 1 \cdot 1 + 1 \cdot 1 - 1.5) + 0.5) = \text{False}$
-1	1	-1	$\phi(1 \cdot \phi(1 \cdot (-1) + 1 \cdot 1 - 1 \cdot (-1) - 2.5) + 1 \cdot \phi(0 \cdot (-1) - 1 \cdot 1 + 1 \cdot (-1) - 1.5) + 0.5) = \text{False}$
-1	-1	1	$\phi(1 \cdot \phi(1 \cdot (-1) + 1 \cdot (-1) - 1 \cdot 1 - 2.5) + 1 \cdot \phi(0 \cdot (-1) - 1 \cdot (-1) + 1 \cdot 1 - 1.5) + 0.5) = \text{True}$
-1	-1	-1	$\phi(1 \cdot \phi(1 \cdot (-1) + 1 \cdot (-1) - 1 \cdot (-1) - 2.5) + 1 \cdot \phi(0 \cdot (-1) - 1 \cdot (-1) + 1 \cdot (-1) - 1.5) + 0.5) = \text{False}$

The 2 tables match

2 Exercise 2

2.1 Point a)

Name all the parameters of the following neural network, together with their dimensions

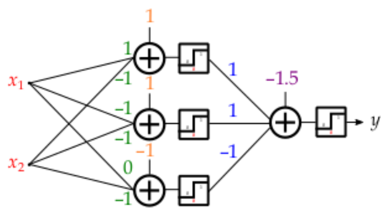

Figure 2: Provided Neural Network

The neural network has the following parameters:

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

This is the **inputs** vector, a 2x1 column vector

$$W = \begin{pmatrix} 1 & -1 \\ -1 & -1 \\ 0 & -1 \end{pmatrix}$$

First layer weights matrix, which is 3x2

$$b = \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix}$$

This is the vector of the biases of the first layer, one for each neuron, so it is a 3x1 column vector.

$$U = \begin{pmatrix} 1 & 1 & -1 \end{pmatrix}$$

Matrix of the weights of the second layer. 1x3 column vector (1 neuron, 3 input)

$$c = -1.5$$

The final bias is a scalar

2.2 Point b)

$$\text{y} = \text{step}\left(\begin{pmatrix} 1 & 1 & -1 \end{pmatrix} \cdot \text{step}\left(\left(\begin{pmatrix} 1 & -1 \\ -1 & -1 \\ 0 & -1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} \right) \right) - 1.5 \right)$$

2.3 Point c)

```python
# --------- Importing useful libraries -------
import numpy as np
import matplotlib.pyplot as plt

# --------- Define activation function and neural network layers
def step_fun(x):
    return np.where(x >= 0, 1, -1)

def f_x(inputs, weights_1, b1, weights_2, b2):
    x1, x2 = inputs
    inputs = np.array([x1, x2]).reshape(-1,1)
    first = np.add(np.matmul(weights_1, inputs),b1)
    first = step_fun(first)
    second = np.add(np.matmul(weights_2, first), b2 )
    res = step_fun(second)

    return res

# create 1000 random points

x = np.random.uniform(-2, 2, size=(1000, 2))

# --------- define neural network parameters ---------

w = np.array([[1, -1], [-1, -1], [0, -1]]) # weights matrix
b = np.array([[1], [1], [-1]])
```

```
u = np.array([1, 1, -1])
c = -1.5

x1_list = []
x2_list = []
colors = []

# --------- For each input (x1, x2), compute the output

for tpl in x:
    x1, x2 = tpl
    res = f_x(tpl, w, b, u, c)
    print(f'Result: {res}')
    x1_list.append(x1)
    x2_list.append(x2)
    if res == 1:
        colors.append('red')
    else:
        colors.append('blue')

# --------- create plot --------
plt.figure(figsize = (8,8))
plt.scatter(x1_list, x2_list, c = colors)
plt.grid()
plt.xlabel('x1')
plt.ylabel('x2')
plt.title('Data points Scatter Plot')

plt.scatter([], [], c='red', label='Result = 1')
plt.scatter([], [], c='blue', label='Result = 0')
plt.legend()  # Add legend

#plt.show()
plt.savefig('scatterplot.png')
```

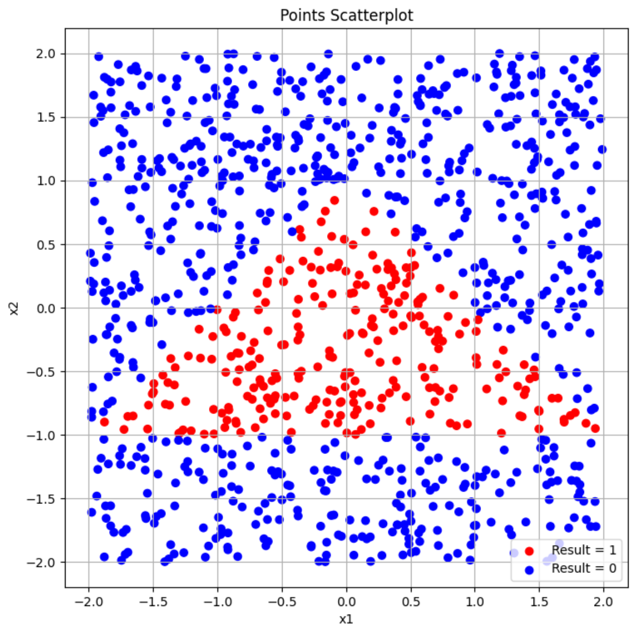

Figure 3: Obtained scatterplot, representing the distribution of the results. In red are all the points that yielded '1' as a result, in blue those which yielded '0'

2.4 Point d)

Compute and draw the decision boundary

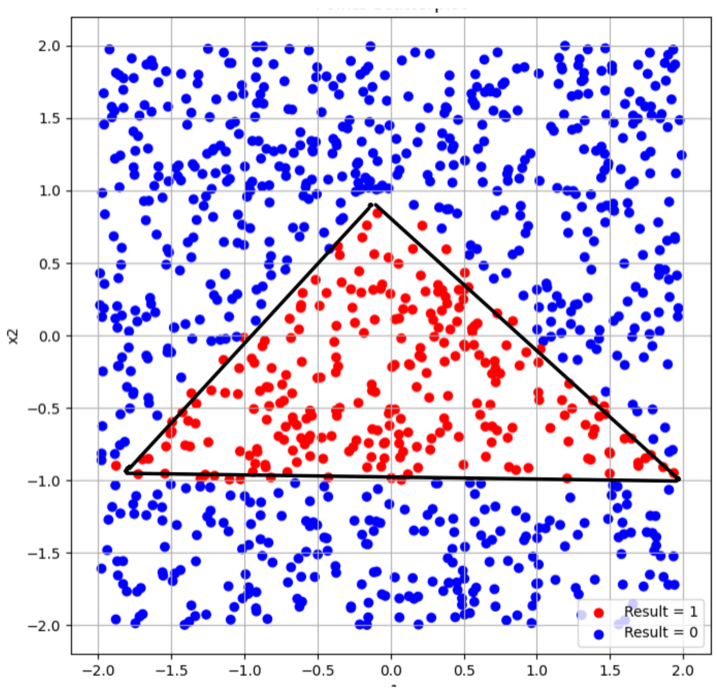

Figure 4:

References

[1] Numpy `sign()` function. Available: https://numpy.org/doc/stable/reference/generated/numpy.sign.html

[2] Overleaf: Code Listing with Listings Package. Available: https://www.overleaf.com/learn/latex/Code_listing_with_listings

[3] Matplotlib: Scatterplots in Python. Available: https://matplotlib.org/stable/gallery/shapes_and_collections/scatter.html

[4] Decision Boundary. Available: https://stackoverflow.com/questions/54399055/plotting-decision-boundary-for-a-neural-network-with-two-layers

[5] Decision Boundary Pt 2. Available: https://psrivasin.medium.com/plotting-decision-boundaries-using-numpy-and-matplotlib-f5613d8acd19

[6] Tables generation aid Available: https://www.tablesgenerator.com/