CO542- NEURAL NETWORKS AND FUZZY SYSTEMS

INTRODUCTION TO ARTIFICIAL NEURAL NETWORKS

F.S.MARZOOK

E/16/232

LAB 03

09/10/2021

Exercise 1

1.

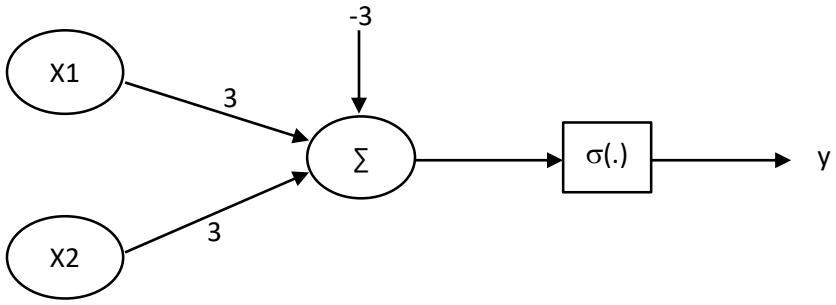

Fig: 1.1 Single-layer Perceptron Model for AND Gate

X1	X2	σ(3X1 + 3X2 - 3)
0	0	σ(3*0 + 3*0 - 3) ≈ 0
0	1	σ(3*0 + 3*1 - 3) ≈ 0
1	0	σ(3*1 + 3*0 - 3) ≈ 0
1	1	σ(3*1 + 3*1 - 3) ≈ 1

The output is an AND gate. Therefore, it is capable of a single layer perceptron to model the AND gate.

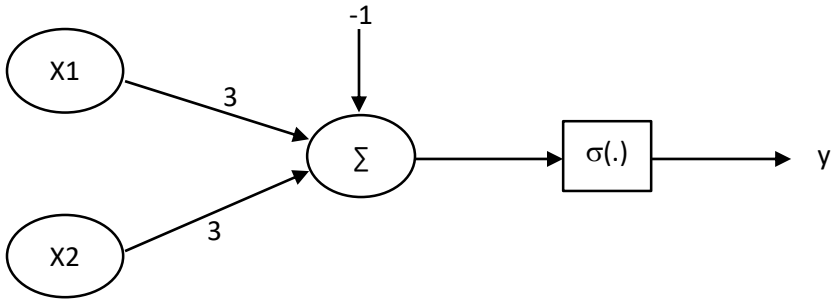

Fig: 1.2 Single-layer Perceptron Model for OR Gate

X1	X2	σ(3X1 + 3X2 - 1)
0	0	σ(3*0 + 3*0 - 1) ≈ 0
0	1	σ(3*0 + 3*1 - 1) ≈ 1
1	0	σ(3*1 + 3*0 - 1) ≈ 1
1	1	σ(3*1 + 3*1 - 1) ≈ 1

The output is an OR gate. Therefore, it is capable of a single layer perceptron to model the OR gate.

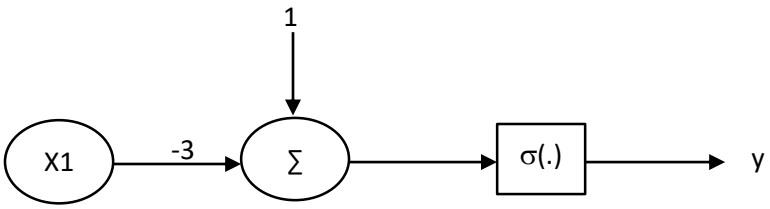

Fig: 1.3 Single-layer Perceptron Model for NOT Gate

X1	σ(-3X1 + 1)
0	σ(-3*0 + 1) ≈ 1
1	σ(-3*1 + 1) ≈ 0

The output is an NOT gate. Therefore, it is capable of a single layer perceptron to model the NOT gate.

2.

```
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import numpy as np
from itertools import product

# Creating a variable named data that is a list that contains the four possible inputs to an AND gate.
data = [[0, 0], [0, 1], [1, 0], [1, 1]]
labels = [0, 0, 0, 1]

# plt.scatter([point[0] for point in data], [point[1] for point in data], c=labels)
# The third parameter "c = labels" will make the points with label 1 a different color than points with label 0.
# plt.show()

# Building a perceptron to learn AND.
classifier = Perceptron(max_iter=40)
classifier.fit(data, labels)
print(classifier.score(data, labels))

x_test = np.arange(0, 1, 0.009)
y_test = np.arange(0, 1, 0.009)

X, Y = np.meshgrid(x_test, y_test)
z_test = np.array([X, Y]).T.reshape(-1, 2).tolist()

predictions = classifier.predict(z_test)
predictions = predictions.reshape(x_test.size, y_test.size)
print('Prediction:')
print(predictions)

# Plot the surface.
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(X, Y, predictions, cmap='plasma', rstride=1, cstride=1)
ax.set_title('Surface plot of AND Gate')
ax.set_xlabel('Input - 1')
ax.set_ylabel('Input - 2')
plt.show()
```

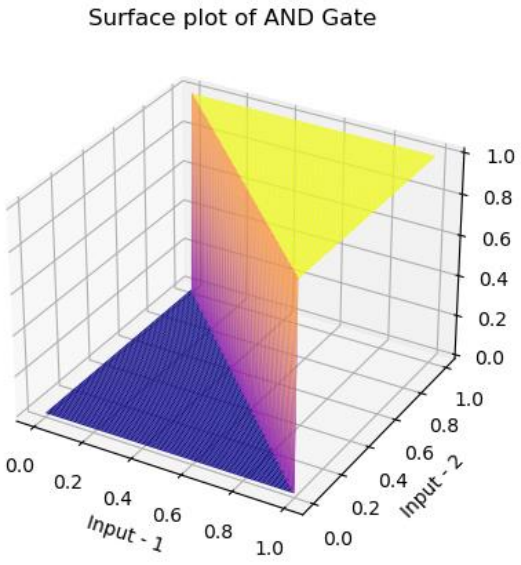

Fig: 1.4 Surface Plot for AND Gate

```python
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import numpy as np
from itertools import product

# Creating a variable named data that is a list that contains the four possible inputs to an OR gate.
data = [[0, 0], [0, 1], [1, 0], [1, 1]]
labels = [0, 1, 1, 1]

# plt.scatter([point[0] for point in data], [point[1] for point in data], c=labels)
# The third parameter "c = labels" will make the points with label 1 a different color than points with label 0.
# plt.show()

# Building a perceptron to learn OR.
classifier = Perceptron(max_iter=40)
classifier.fit(data, labels)
print(classifier.score(data, labels))

x_test = np.arange(0, 1, 0.009)
y_test = np.arange(0, 1, 0.009)

X, Y = np.meshgrid(x_test, y_test)
z_test = np.array([X, Y]).T.reshape(-1, 2).tolist()

predictions = classifier.predict(z_test)
predictions = predictions.reshape(x_test.size, y_test.size)
print('Prediction:')
print(predictions)

# Plot the surface.
fig, ax = plt.subplots(subplot_kw={"projection": "3d"})
surf = ax.plot_surface(X, Y, predictions, cmap='plasma', rstride=1, cstride=1)
ax.set_title('Surface plot of OR Gate')
ax.set_xlabel('Input - 1')
ax.set_ylabel('Input - 2')
plt.show()
```

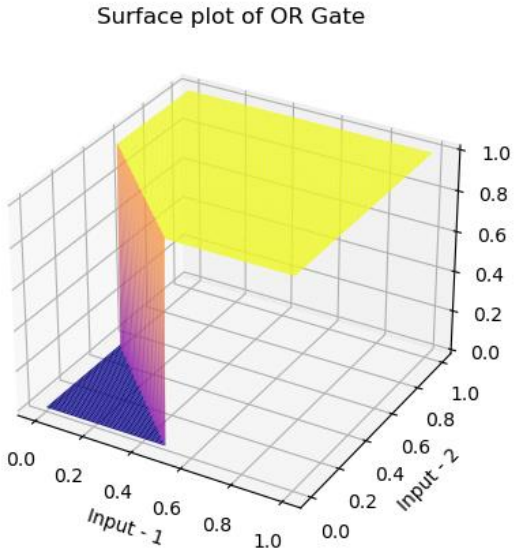

Fig: 1.5 Surface Plot for OR Gate

```
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import numpy as np
from itertools import product

# Creating a variable named data that is a list that contains
# the four possible inputs to an NOT gate.
data = [[0], [1]]
labels = [1, 0]

# Building a perceptron to learn NOT.
classifier = Perceptron(max_iter=40)
classifier.fit(data, labels)
print(classifier.score(data, labels))

x_test = np.arange(0, 1, 0.009)
x_test = x_test.reshape(-1, 1)

predictions = classifier.predict(x_test)
print('Prediction:')
print(predictions)

# Plot the surface.
plt.plot(x_test, predictions)
plt.title('Surface plot of NOT Gate')
plt.xlabel('Input')
plt.ylabel('Prediction')
plt.show()
```

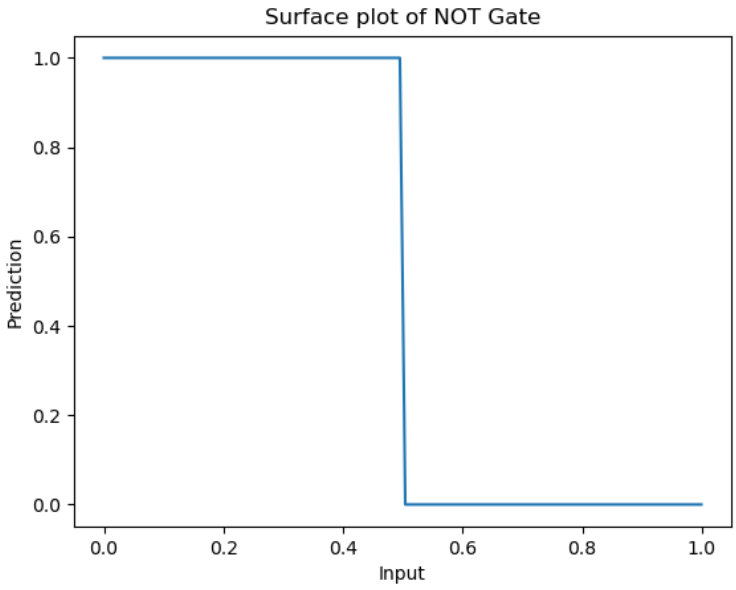

Fig: 1.6 Surface Plot for NOT Gate

Exercise 2

1.

```
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import time

data = [[0.1, 0.1], [0.8, 0.3], [0.3, 0.4], [0.6, 0.9]]
labels = [0, 1, 0, 1]

# Plot these vectors (Use a scatter plot)
plt.scatter([point[0] for point in data], [point[1] for point in data], c=labels)
plt.show()

# Building a perceptron
classifier = Perceptron(max_iter=40)

# Observe the training time
start_time = time.time()
classifier.fit(data, labels)
stop_time = time.time()

training_time = stop_time - start_time

print('Training time = ', training_time)
print('No. of iterations = ', classifier.n_iter_)
```

2.

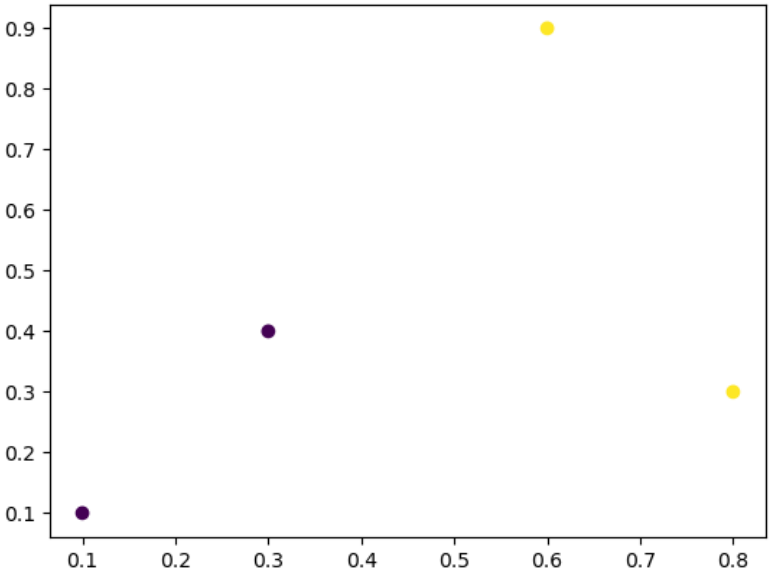

Fig: 2.1 Scatter Plot for the 4 Input Vectors

3. It took 9 iterations for the training.
   The training time = 17.497 ms

4.

```
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import time

data = [[0.1, 0.1], [0.8, 0.3], [0.3, 0.4], [40, 55]]
labels = [0, 1, 0, 1]

# Plot these vectors (Use a scatter plot)
plt.scatter([point[0] for point in data], [point[1] for point in data], c=labels)
plt.show()

# Building a perceptron
classifier = Perceptron(max_iter=40)

# Observe the training time
start_time = time.time()
classifier.fit(data, labels)
stop_time = time.time()

training_time = stop_time - start_time

print('Training time = ', training_time)
print('No. of iterations = ', classifier.n_iter_)
```

5.

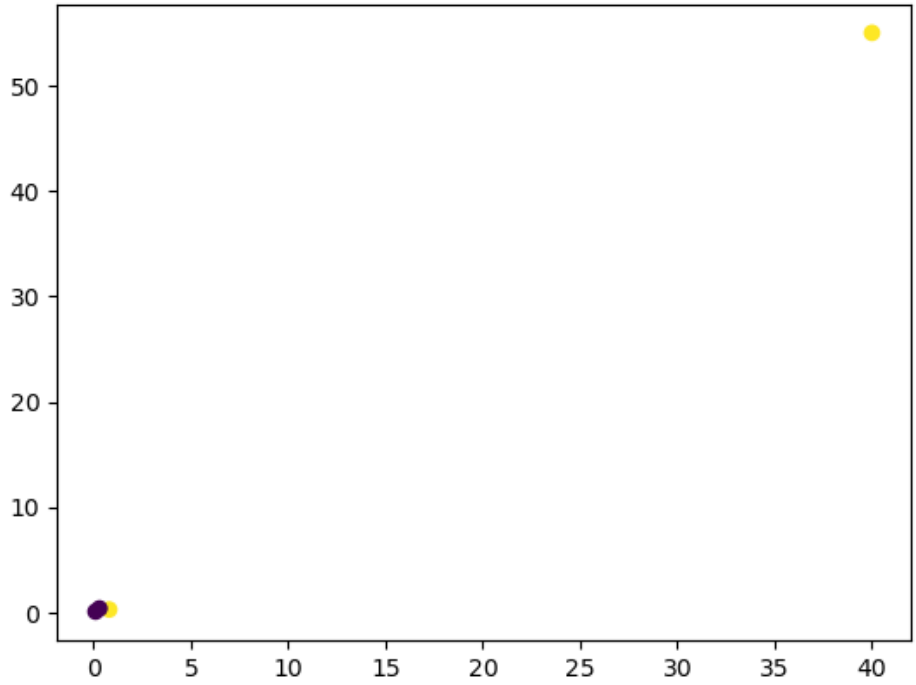

Fig: 2.2 Scatter Plot for the 4 Input Vectors

6. It took 29 iterations for the training.
   The training time = 21.960 ms

   In exercise 2.3
   No. of iterations = 9
   Training time = 17.497 ms

   From these results, we can say that the training time calculated in exercise 2.3 for inputs with smaller magnitudes is a little bit faster than the other. But when one input vector is changed as much larger than all the other vectors the number of iterations increases.

7-9.

```python
from sklearn.linear_model import Perceptron
import matplotlib.pyplot as plt
import time
import numpy as np
from mlxtend.plotting import plot_decision_regions

data = [[0.1, 0.1], [0.8, 0.3], [0.3, 0.4], [0.6, 0.9]]
labels = [0, 1, 0, 1]

# Plot these vectors (Use a scatter plot)
plt.scatter([point[0] for point in data], [point[1] for point in data], c=labels)
plt.show()

# Building a perceptron
classifier = Perceptron(max_iter=40)
# Observe the training time
start_time = time.time()
classifier.fit(data, labels)
stop_time = time.time()

training_time = stop_time - start_time

print('Training time = ', training_time)
print('No. of iterations = ', classifier.n_iter_)

# test input
x_test = [[0.5, 0.7]]
prediction = classifier.predict(x_test)
print('Prediction: ', prediction)

data = np.array(data)
labels = np.array(labels)
plot_decision_regions(data, labels, clf=classifier, legend=2)       # Plotting the classification line
# Adding the test input to the plot
plt.scatter([point[0] for point in x_test], [point[1] for point in x_test], label='test input', c='g')
plt.axis([0, 0.9, 0, 1])               # Zoom in the plot
plt.show()
```

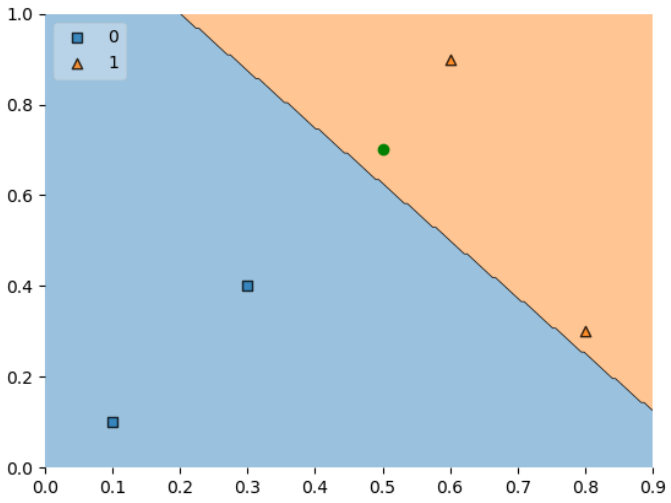

Fig: 2.3 Plot for Classification line

Exercise 3

```
import os
from skimage import io
import numpy as np
from sklearn.linear_model import Perceptron

MAIN_PATH_TRAIN = r"C:\Users\user\Desktop\CO542\Lab3\Exercise3\classpics\train"
MAIN_PATH_TEST = r"C:\Users\user\Desktop\CO542\Lab3\Exercise3\classpics\test"
SUBSETS = ["brown_hair", "black_hair"]

features_train = []
target_train = []
for subset in SUBSETS:
    full_path = os.path.join(MAIN_PATH_TRAIN, subset)
    full_path_listdir = os.listdir(full_path)
    for img_name in full_path_listdir:
        img_path = os.path.join(full_path, img_name)
        image = io.imread(img_path)
        image = (np.array(image.ravel()))
        image = np.array(image)
        features_train.append(image)
        target_train.append(subset)

features_test = []
target_test = []
for subset in SUBSETS:
    full_path = os.path.join(MAIN_PATH_TEST, subset)
    full_path_listdir = os.listdir(full_path)
    for img_name in full_path_listdir:
        img_path = os.path.join(full_path, img_name)
        image = io.imread(img_path)
        image = (np.array(image.ravel()))
        image = np.array(image)
        features_test.append(image)
        target_test.append(subset)

# Building a perceptron model and train the data
classifier = Perceptron(max_iter=40)
classifier.fit(np.array(features_train), np.array(target_train))
print(classifier.score(np.array(features_train), np.array(target_train)))
predictions = classifier.predict(np.array(features_test))

print('Prediction:')
print(predictions)
print('Actual:')
print(np.array(target_test))
```

Here, students were grouped as,

- People with Brown Hair:

- People with Black Hair:

```
Prediction:
['brown_hair' 'brown_hair' 'brown_hair' 'black_hair' 'black_hair'
 'black_hair' 'black_hair']
Actual:
['brown_hair' 'brown_hair' 'brown_hair' 'black_hair' 'black_hair'
 'black_hair' 'black_hair']
```

Fig: 3.1 Output of the Actual and the Predicted Value

This model exactly predicts the given test data correctly.