Σειρά προβλημάτων 3

Κυρίτσης Βασίλειος 02999 και Σταμούλος Αλέξανδρος 02954

Νευρο-Ασαφής Υπολογιστική 2023-24
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας, Βόλος
{vakyritsis, astamoulos}@e-ce.uth.gr

1 Problem-01

Από την εκφώνηση προβλήματος, γνωρίζουμε ότι το δίκτυο θα πρέπει να έχει δύο εισόδους και μπορούμε να χρησιμοποιήσουμε μία έξοδο για να διακρίνουμε τις δύο κλάσεις. Επιλέγουμε θετική έξοδο για διανύσματα Κατηγορίας I και αρνητική έξοδο για Διανύσματα κατηγορίας II. Η περιοχή Κατηγορίας I αποτελείται από δύο απλές υποπεριοχές, και φαίνεται ότι δύο νευρώνες θα πρέπει να είναι επαρκείς για την εκτέλεση της ταξινόμησης. Οι σειρές του πίνακα βαρών του πρώτου layer θα δημιουργήσουν κέντρα για τις δύο βασικές συναρτήσεις και θα επιλέξουμε κάθε κέντρο να βρίσκεται στη μέση μιας υποπεριοχής. Επικεντρώνοντας μια συνάρτηση βάσης σε κάθε υποπεριοχή, μπορούμε να παράγουμε τις μέγιστες εξόδους δικτύου εκεί. Ο πίνακας βάρους του πρώτου layer είναι τότε:

$$W^1 = \begin{bmatrix} -0.5 & 1.5 \\ 2 & 2 \end{bmatrix}$$

Η επιλογή των biases στο πρώτο layer εξαρτάται από το πλάτος που θέλουμε για κάθε basis function. Για αυτό το πρόβλημα, η πρώτη basis function πρέπει να είναι ευρύτερη από τη δεύτερη. Επομένως, το πρώτο bias θα είναι μικρότερο από το δεύτερο bias. Το όριο που σχηματίζεται από την πρώτη basis function θα πρέπει να έχει ακτίνα περίπου 1/2, ενώ η δεύτερη πρέπει να έχει ακτίνα περίπου 1/4. Θέλουμε οι basis function να πέφτουν σημαντικά από τα peak τους σε αυτές τις αποστάσεις. Αν χρησιμοποιήσουμε ένα bias 2 για τον πρώτο νευρώνα και ένα bias 4 για τον δεύτερο νευρώνα, παίρνουμε τα παρακάτω reductions σε απόσταση μίας ακτίνας από τα κέντρα:

$$a = e^{-n^2} = e^{-(2\cdot\frac{1}{2})^2} = e^{-1} = 0.3679, \quad a = e^{-n^2} = e^{-(4\cdot\frac{1}{4})^2} = e^{-1} = 0.3679$$

Αυτές οι τιμές μας κάνουν για το προβλημά μας, άρα διαλέγουμε:

$$b^1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

Η αρχική basis function έχει αποκρίσεις από 0 εώς 1. Θέλουμε η έξοδος να είναι αρνητική για εισόδους έκτος των σημαδεμένων περιοχών, συνεπώς το bias για το δεύτερο layer θα είναι -1 και θα χρησιμοποιήσουμε τιμή 2 για τα βάρη του δεύτερου layer, έτσι ώστε να επαναφέρουμε τα peaks πίσω σε 1. Άρα για το δεύτερο layer έχουμε:

$$W^2 = \begin{bmatrix} 2 & 2 \end{bmatrix}, b^2 = \begin{bmatrix} -1 \end{bmatrix}.$$

2 Problem-02

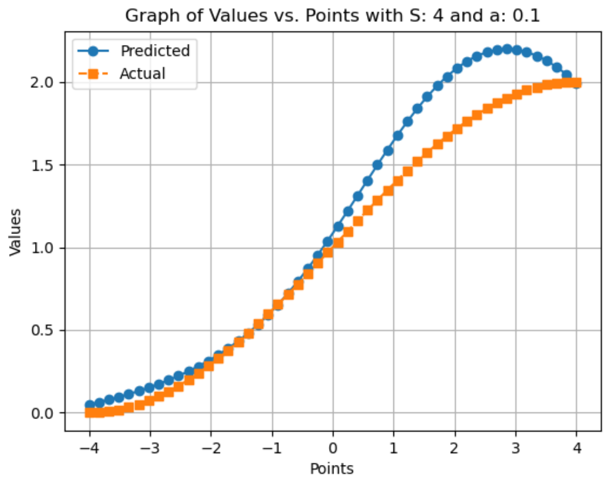
```
import numpy as np
import random
import matplotlib.pyplot as plt
from math import sin, pi, exp, sqrt

# Function to approximate
def g(x):
    return 1 + sin(x * (pi) / 8)

# Activation function of 1st layer and the derivative
def radbas(n):
    return exp(-n*n)

def radbas_der(n):
    return -2*n*exp(-n*n)

# Activation function of 2st layer and the derivative
def purelin(n):
    return n

def purelin_der(n):
    return 1

def training(points, S, learning_rate, w1, b1, w2, b2, sumOfSqrError):
    for i in range(len(points)):
        n1 = []
        a1 = []
        n2 = b2

        for j in range(S):
            n = sqrt((points[i]-w1[j])*(points[i]-w1[j]))*b1[j]
```

```
            n1.append(n)
            a = radbas(n)
            a1.append(a)
            n2 += a * w2[j]

        a2 = purelin(n2)

        # Calculate error
        e = g(points[i]) - a2
        sumOfSqrError += e*e

        # Calculate sensitivities and update weights and biases
        s2 = -2*purelin_der(n2)*(e)
        s1 = []
        for j in range(S):
            s1.append(radbas_der(n1[j])*w2[j]*s2)
            w2[j] -= learning_rate*s2*a1[j]

        b2 -= learning_rate*s2

        for j in range(S):
            w1[j] -= learning_rate*s1[j]*points[i]
            b1[j] -= learning_rate*s1[j]


    return  w1, b1, w2, b2, sumOfSqrError

def plot_response(interval, responses, actual, S, learning_rate):
    plt.plot(interval, responses, marker='o', linestyle='-', label='Predicted')
    plt.plot(interval, actual, marker='s', linestyle='--', label='Actual')
    plt.xlabel('Points')
    plt.ylabel('Values')
    plt.title(f'Graph of Values vs. Points with S: {S} and a: {learning_rate}')
    plt.grid(True)
    plt.legend()
    plt.show()

def feedforward(point, S, w1, b1, w2, b2):
    n1 = []
    a1 = []
    n2 = b2
```

```
    for j in range(S):
        n = sqrt((point-w1[j])*(point-w1[j]))*b1[j]
        n1.append(n)
        a = radbas(n)
        a1.append(a)
        n2 += a * w2[j]

    a2 = purelin(n2)
    actual = g(point)
    return a2, actual

if __name__ == "__main__":
    # Number of random points
    num_points = 30

    # Generate 30 random points within the interval [-4, 4]
    points = [random.uniform(-4, 4) for _ in range(num_points)]
    points.sort()
    # Hyperparameters

    learning_rate = 0.1
    S = 4 # number of neurons, values are 4, 8, 12, 20

    # Weights and bias

    w1 = []
    b1 = []
    w2 = []

    for i in range(S):
        w1.append(random.uniform(-0.5, 0.5))
        b1.append(random.uniform(-0.5, 0.5))
        w2.append(random.uniform(-0.5, 0.5))
    b2 = random.uniform(-0.5, 0.5)

    sumOfSqrError = 0
    # Start training

    epochs = 6
```

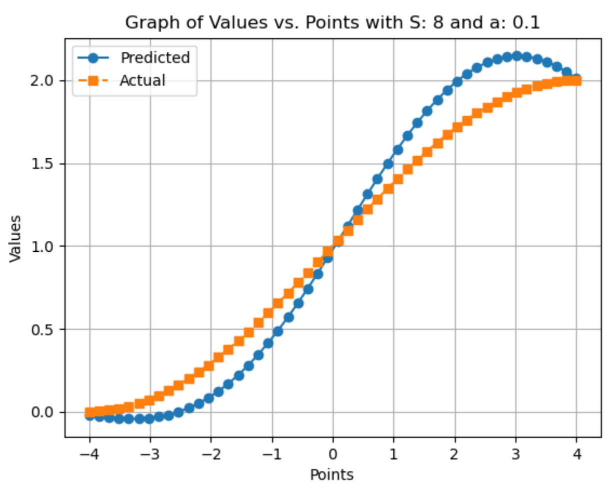
```
for epoch in range(epochs):
    w1, b1, w2, b2, sumOfSqrError = training(points, S, learning_rate, w1, b1,
        w2, b2, sumOfSqrError)


print (f"Final w1: {w1}")
print (f"Final b1: {b1}")
print (f"Final w2: {w2}")
print (f"Final b2: {b2}")
print (f"Sum of squared error over the training set: {sumOfSqrError} ")

# Feed forward for values -4<p<4 in order to plot the network response after training
interval = np.linspace(-4, 4, 50)
responses = []
actual_values = []
for point in interval:
    response, actual = feedforward(point, S, w1, b1, w2, b2)
    responses.append(response)
    actual_values.append(actual)

plot_response(interval, responses, actual_values, S, learning_rate)
```

Αρχικά για την εκπαίδευση του δικτύου χρησιμοποιήσαμε τα δεδομένα εισόδου για 6 epochs, καθώς με τους πειραματισμούς μας καταλήξαμε ότι είναι ένα ικανοποιητικό νούμερο. Για κάθε S^1 δοκιμάσαμε διαφορικές τιμές για το learning rate.
<u>a = 0.1:</u>



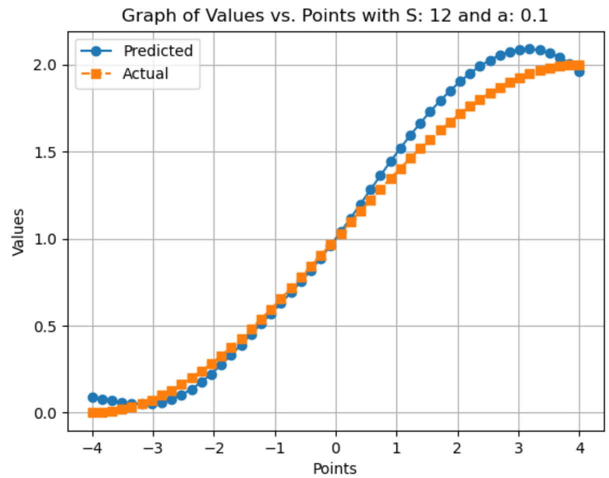
Σχήμα 1. s=4, MSE= 10.4

[H]



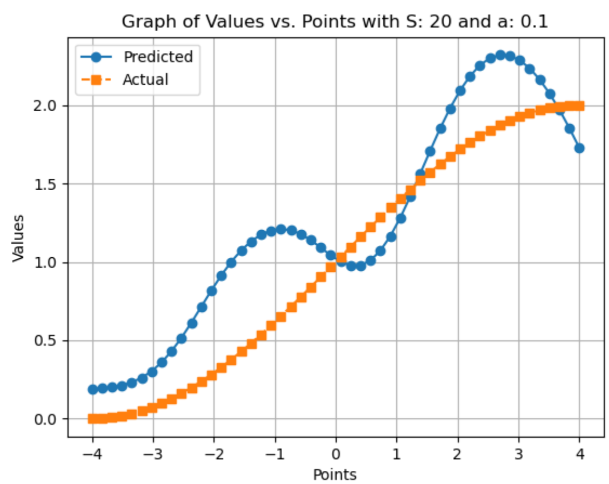
Σχήμα 2. s=8, MSE= 11.9

[H]



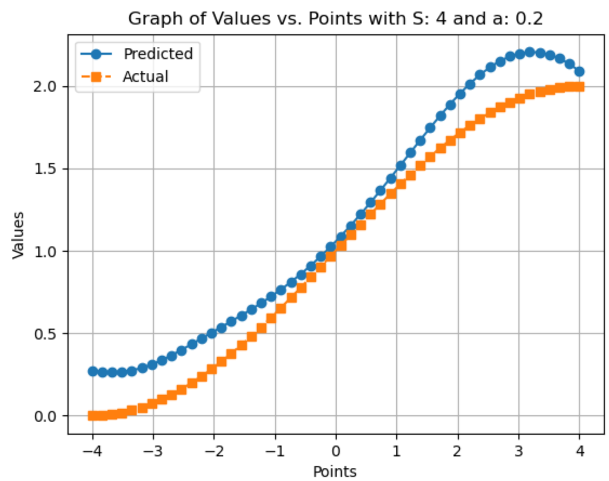
Σχήμα 3. s=12, MSE= 49.6

[H]



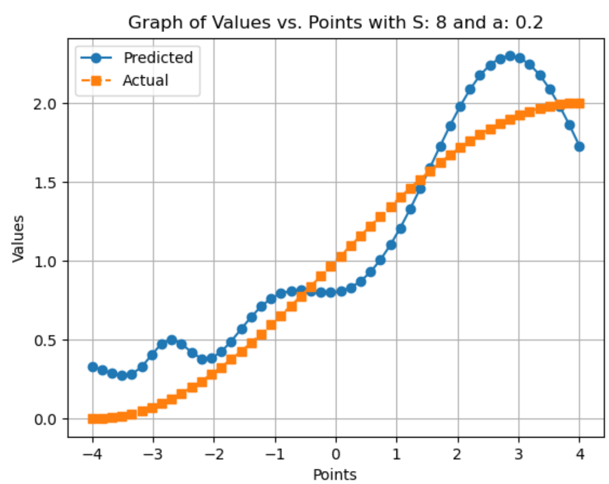
Σχήμα 4. s=20, MSE= 89.673

<u>a = 0.2:</u>



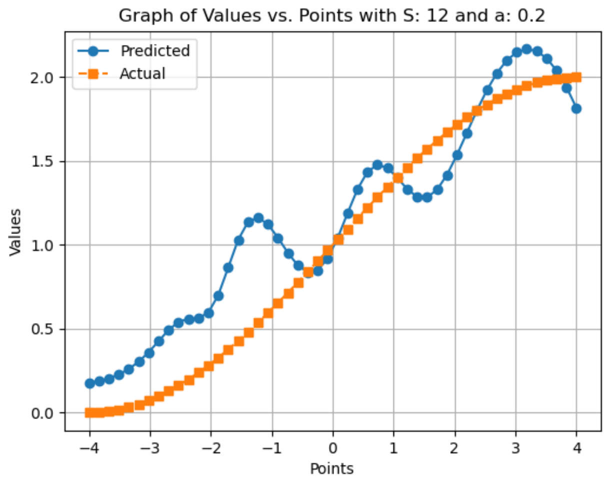
Σχήμα 5. s=4, MSE= 13.01

[H]



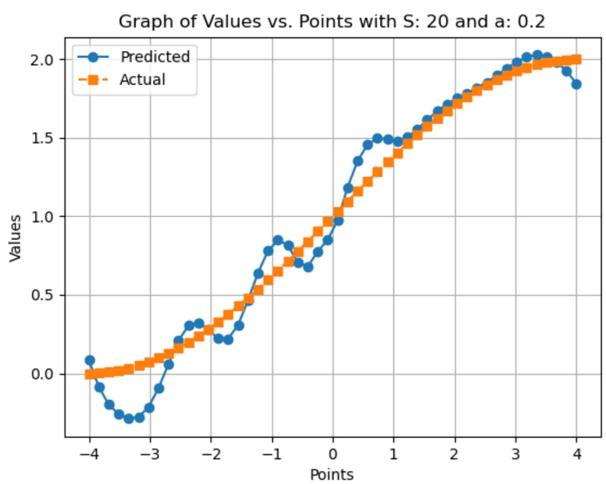
Σχήμα 6. s=8, MSE= 57.355

[H]



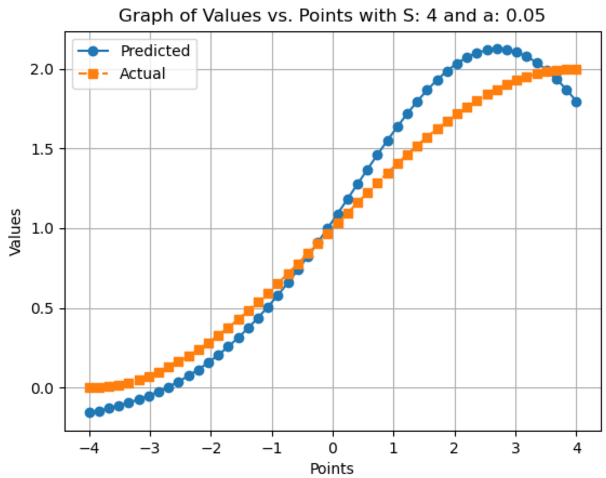
Σχήμα 7. s=12, MSE= 76.790

[H]



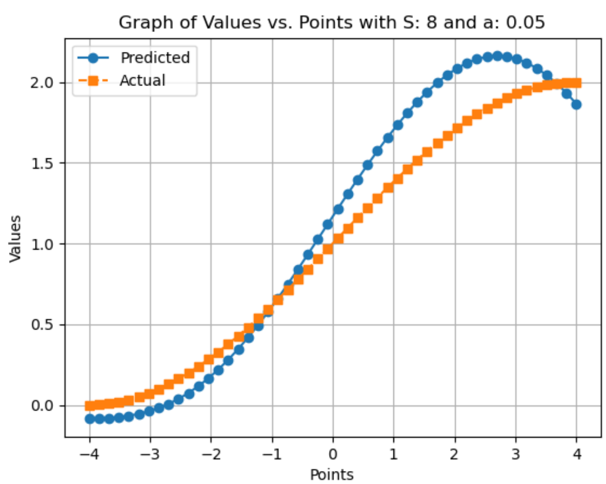
Σχήμα 8. s=20, MSE= 138.509

<u>a = 0.05:</u>



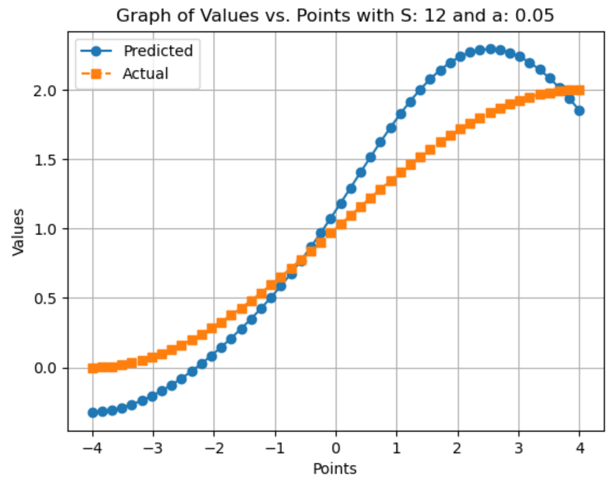
Σχήμα 9. s=4, MSE= 17.591

[H]



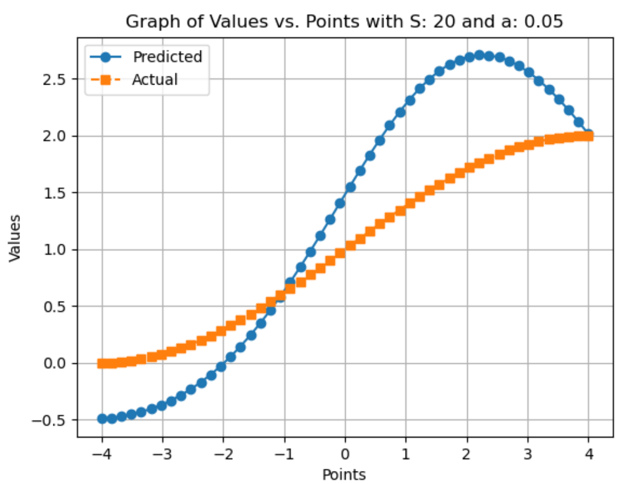
Σχήμα 10. s=8, MSE= 16.560

[H]



Σχήμα 11. s=12, MSE= 13.0357

[H]



Σχήμα 12. s=20, MSE= 18.1098

Παρατηρήσεις:

– Το α = 0.1 είναι ένα stable learing rate για το νευρωνικό μας, αφού μας δίνει αποδεκτά αποτελέσματα για οποιονδήποτε αριθμό κέντρων επιλέξουμε.
– Μεγαλύτερες τιμές του α (α = 0.2) μας δίνουν καλύτερα αποτελέσματα όταν μικραίνει ο αριθμός των κέντρων (S = 4, S = 8).
– Για μεγαλύτερο αριθμό κέντρων (S = 12, S = 20) χρειαζόμαστε μικρότερο learning rate (α = 0.05) για να πάρουμε ικανοποιητικά αποτελέσματα.

3 Problem-03

Έχουμε τα παρακάτω διανύσματα και βάρη:

$$p_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, p_2 = \begin{bmatrix} -1 \\ 2 \end{bmatrix}, p_3 = \begin{bmatrix} -2 \\ -2 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, w_2 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Έχουμε

$$W = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Αρχικά ξεκινάμε με το διάνυσμα p_1:

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_1|| \\ -||w_2 - p_1|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -||\begin{bmatrix} 0 & 1 \end{bmatrix}^T - \begin{bmatrix} 1 & 1 \end{bmatrix}^T|| \\ -||\begin{bmatrix} 1 & 0 \end{bmatrix}^T - \begin{bmatrix} 1 & 1 \end{bmatrix}^T|| \end{bmatrix}\Big)$$

$$= compet\Big(\begin{bmatrix} -||\begin{bmatrix} -1 & 0 \end{bmatrix}^T|| \\ -||\begin{bmatrix} 0 & -1 \end{bmatrix}^T|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -1 \\ -1 \end{bmatrix}\Big) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Το w_1 και το w_2 απέχουν την ίδια απόσταση απο το p_1, ωστόσο το w_1 επειδή έχει μικρότερο index κερδίζει το competition και βγάζει output 1. Συνεπώς κερδίζει ο πρώτος νευρώνας και έτσι το βάρος του μπορεί να κινηθεί πιο κοντα στο p_1. Χρησιμοποιούμε τον Kohenen rule για να κάνουμε update το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_1 - w_{1_{old}}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 1 \end{bmatrix}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} 1 \\ 0 \end{bmatrix}) = \begin{bmatrix} 0 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$$

$$w_{1_{new}} = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}$$

p_2:

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_2|| \\ -||w_2 - p_2|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -|| \begin{bmatrix} 0.5 & 1 \end{bmatrix}^T - \begin{bmatrix} -1 & 2 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 1 & 0 \end{bmatrix}^T - \begin{bmatrix} -1 & 2 \end{bmatrix}^T || \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -|| \begin{bmatrix} 1.5 & -1 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 2 & -2 \end{bmatrix}^T || \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -1.80278 \\ -2.82842 \end{bmatrix}\Big) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_2 - w_{1_{old}}) = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} -1 \\ 2 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 1 \end{bmatrix}) = \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} -1.5 \\ 1 \end{bmatrix})$$
$$= \begin{bmatrix} 0.5 \\ 1 \end{bmatrix} + \begin{bmatrix} -0.75 \\ 0.5 \end{bmatrix} = \begin{bmatrix} -0.25 \\ 1.5 \end{bmatrix}$$

p_3 :

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_3|| \\ -||w_2 - p_3|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -|| \begin{bmatrix} -0.25 & 1.5 \end{bmatrix}^T - \begin{bmatrix} -2 & -2 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 1 & 0 \end{bmatrix}^T - \begin{bmatrix} -2 & -2 \end{bmatrix}^T || \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -|| \begin{bmatrix} 1.75 & 3.5 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 3 & 2 \end{bmatrix}^T || \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -3.91312 \\ -3.6055\sqrt{2} \end{bmatrix}\Big) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Κερδίζει ο δεύτερος νευρώνας άρα αλλάζουμε το w_2:

$$w_{2_{new}} = w_{2_{old}} + \alpha(p_3 - w_{2_{old}}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.5(\begin{bmatrix} -2 \\ -2 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \end{bmatrix}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + 0.5(\begin{bmatrix} -3 \\ -2 \end{bmatrix}) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -1.5 \\ -1 \end{bmatrix}$$
$$w_{2_{new}} = \begin{bmatrix} -0.5 \\ -1 \end{bmatrix}$$

p_2 :

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_2|| \\ -||w_2 - p_2|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -|| \begin{bmatrix} -0.25 & 1.5 \end{bmatrix}^T - \begin{bmatrix} -1 & 2 \end{bmatrix}^T || \\ -|| \begin{bmatrix} -0.5 & -1 \end{bmatrix}^T - \begin{bmatrix} -1 & 2 \end{bmatrix}^T || \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -|| \begin{bmatrix} 0.75 & -0.5 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 0.5 & -3 \end{bmatrix}^T || \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -0.901388 \\ -3.04138 \end{bmatrix}\Big) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_2 - w_{1_{old}}) = \begin{bmatrix} -0.25 \\ 1.5 \end{bmatrix} + 0.5(\begin{bmatrix} -1 \\ 2 \end{bmatrix} - \begin{bmatrix} -0.25 \\ 1.5 \end{bmatrix}) = \begin{bmatrix} -0.25 \\ 1.5 \end{bmatrix} + 0.5(\begin{bmatrix} -0.75 \\ 0.5 \end{bmatrix})$$
$$= \begin{bmatrix} -0.25 \\ 1.5 \end{bmatrix} + \begin{bmatrix} -0.375 \\ 0.25 \end{bmatrix} = \begin{bmatrix} -0.625 \\ 1.75 \end{bmatrix}$$

p_3 :

$$a^1 = compet(n^1) = compet\left(\begin{bmatrix} -||w_1 - p_3|| \\ -||w_2 - p_3|| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -|| \begin{bmatrix} -0.625 & 1.75 \end{bmatrix}^T - \begin{bmatrix} -2 & -2 \end{bmatrix}^T || \\ -|| \begin{bmatrix} -0.5 & -1 \end{bmatrix}^T - \begin{bmatrix} -2 & -2 \end{bmatrix}^T || \end{bmatrix}\right)$$
$$= compet\left(\begin{bmatrix} -|| \begin{bmatrix} 1.375 & 3.75 \end{bmatrix}^T || \\ -|| \begin{bmatrix} 1.5 & 1 \end{bmatrix}^T || \end{bmatrix}\right) = compet\left(\begin{bmatrix} -3.99414 \\ -1.80278 \end{bmatrix}\right) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Κερδίζει ο δεύτερος νευρώνας άρα αλλάζουμε το w_2:

$$w_{2_{new}} = w_{2_{old}} + \alpha(p_3 - w_{2_{old}}) = \begin{bmatrix} -0.5 \\ -1 \end{bmatrix} + 0.5(\begin{bmatrix} -2 \\ -2 \end{bmatrix} - \begin{bmatrix} -0.5 \\ -1 \end{bmatrix}) = \begin{bmatrix} -0.5 \\ -1 \end{bmatrix} + 0.5(\begin{bmatrix} -1.5 \\ -1 \end{bmatrix})$$
$$= \begin{bmatrix} -0.5 \\ -1 \end{bmatrix} + \begin{bmatrix} -0.75 \\ -0.5 \end{bmatrix} = \begin{bmatrix} -1.25 \\ -1.5 \end{bmatrix}$$

p_1 :

$$a^1 = compet(n^1) = compet\left(\begin{bmatrix} -||w_1 - p_1|| \\ -||w_2 - p_1|| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -|| \begin{bmatrix} -0.625 & 1.75 \end{bmatrix}^T - \begin{bmatrix} 1 & 1 \end{bmatrix}^T || \\ -|| \begin{bmatrix} -1.25 & -1.5 \end{bmatrix}^T - \begin{bmatrix} 1 & 1 \end{bmatrix}^T || \end{bmatrix}\right)$$
$$= compet\left(\begin{bmatrix} -|| \begin{bmatrix} -1.625 & 0.75 \end{bmatrix}^T || \\ -|| \begin{bmatrix} -2.25 & -2.5 \end{bmatrix}^T || \end{bmatrix}\right) = compet\left(\begin{bmatrix} -1.78973 \\ -3.36341 \end{bmatrix}\right) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_1 - w_{1_{old}}) = \begin{bmatrix} -0.625 \\ 1.75 \end{bmatrix} + 0.5(\begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} -0.625 \\ 1.75 \end{bmatrix}) = \begin{bmatrix} -0.625 \\ 1.75 \end{bmatrix} + 0.5(\begin{bmatrix} 1.625 \\ -0.75 \end{bmatrix})$$
$$= \begin{bmatrix} -0.625 \\ 1.75 \end{bmatrix} + \begin{bmatrix} 0.8125 \\ -0.375 \end{bmatrix} = \begin{bmatrix} 0.1875 \\ 1.375 \end{bmatrix}$$

Συνεπώς μετά από την εκπαίδευση του δικτύου οι τελικές τιμές των βαρών είναι:

$$w_1 = \begin{bmatrix} 0.1875 \\ 1.375 \end{bmatrix}, w_2 = \begin{bmatrix} -1.25 \\ -1.5 \end{bmatrix}$$

Παρακάτω παραθέτουμε τον κώδικα μας για το πρόβλημα:

```python
import math

def competitive_layer(input_list):
    """
```

```
    Competitive layer that returns the index of the biggest number,
    with the smallest index breaking ties.
    """
    max_index = 0
    max_value = input_list[0]

    for i in range(1, len(input_list)):
        if input_list[i] > max_value:
            max_index = i
            max_value = input_list[i]
        elif input_list[i] == max_value:
            max_index = min(max_index, i)

    result = [0] * len(input_list)
    result[max_index] = 1

    return result

# Kohonen Rule for update
def update_weights(a1, W, learning_rate, p):
    index = a1.index(1)

    difference = [p[i] - W[index][i] for i in range(len(p))]
    product = [learning_rate * element for element in difference]
    W[index] = [W[index][i] + product[i] for i in range(len(product))]

    return W

def iteration(p, W, learning_rate):
    n1 = []
    for i in range(0, len(W)):
        n1.append(-math.sqrt((W[i][0] - p[0])**2 + (W[i][1] - p[1])**2))

    a1  = competitive_layer(n1)

    W = update_weights(a1, W, learning_rate, p)

    return W
if __name__ == "__main__":

    # Init points and weights
```

```
p1 = [1, 1]
p2 = [-1, 2]
p3 = [-2, -2]

w1 = [0, 1]
w2 = [1, 0]

W = [w1, w2]

# Learning rate
learning_rate = 0.5

training_set = [p1, p2, p3, p2, p3, p1]

for p in training_set:
    W = iteration(p, W, learning_rate)

print("Weights after trainning: ")
print(W)
```

output :
Weights after trainning: [[0.1875, 1.375], [-1.25, -1.5]]

Τα αποτελέσματα του προγράμματος συμβαδίζουν με αυτά που υπολογίσαμε παραπάνω.

4 Problem-04

Έχουμε τα παρακάτω διανύσματα και βάρη:

$$p_1 = \begin{bmatrix} 2 \\ 0 \end{bmatrix}, p_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}, p_3 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$$

$$w_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, w_2 = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Έχουμε

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

p_1:

$$a^1 = compet(n^1) = compet\left(\begin{bmatrix} -||w_1 - p_1|| \\ -||w_2 - p_1|| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -||\begin{bmatrix}1 & 0\end{bmatrix}^T - \begin{bmatrix}2 & 0\end{bmatrix}^T|| \\ -||\begin{bmatrix}-1 & 0\end{bmatrix}^T - \begin{bmatrix}2 & 0\end{bmatrix}^T|| \end{bmatrix}\right)$$

$$= compet\left(\begin{bmatrix} -||\begin{bmatrix}-1 & 0\end{bmatrix}^T|| \\ -||\begin{bmatrix}-3 & 0\end{bmatrix}^T|| \end{bmatrix}\right) = compet\left(\begin{bmatrix}-1 \\ -3\end{bmatrix}\right) = \begin{bmatrix}1 \\ 0\end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_1 - w_{1_{old}}) = \begin{bmatrix}1 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}2 \\ 0\end{bmatrix} - \begin{bmatrix}1 \\ 0\end{bmatrix}) = \begin{bmatrix}1 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}1 \\ 0\end{bmatrix}) = \begin{bmatrix}1 \\ 0\end{bmatrix} + \begin{bmatrix}0.5 \\ 0\end{bmatrix}$$

$$w_{1_{new}} = \begin{bmatrix}1.5 \\ 0\end{bmatrix}$$

p_2:

$$a^1 = compet(n^1) = compet\left(\begin{bmatrix} -||w_1 - p_2|| \\ -||w_2 - p_2|| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -||\begin{bmatrix}1.5 & 0\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T|| \\ -||\begin{bmatrix}-1 & 0\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T|| \end{bmatrix}\right)$$

$$= compet\left(\begin{bmatrix} -||\begin{bmatrix}1.5 & -1\end{bmatrix}^T|| \\ -||\begin{bmatrix}-1 & -1\end{bmatrix}^T|| \end{bmatrix}\right) = compet\left(\begin{bmatrix}-1.80278 \\ -1.4142\end{bmatrix}\right) = \begin{bmatrix}0 \\ 1\end{bmatrix}$$

Κερδίζει ο δεύτερος νευρώνας άρα αλλάζουμε το w_2:

$$w_{2_{new}} = w_{2_{old}} + \alpha(p_2 - w_{2_{old}}) = \begin{bmatrix}-1 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}0 \\ 1\end{bmatrix} - \begin{bmatrix}-1 \\ 0\end{bmatrix}) = \begin{bmatrix}-1 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}1 \\ 1\end{bmatrix}) = \begin{bmatrix}-1 \\ 0\end{bmatrix} + \begin{bmatrix}0.5 \\ 0.5\end{bmatrix}$$

$$w_{2_{new}} = \begin{bmatrix}-0.5 \\ 0.5\end{bmatrix}$$

p_3:

$$a^1 = compet(n^1) = compet\left(\begin{bmatrix} -||w_1 - p_3|| \\ -||w_2 - p_3|| \end{bmatrix}\right) = compet\left(\begin{bmatrix} -||\begin{bmatrix}1.5 & 0\end{bmatrix}^T - \begin{bmatrix}2 & 2\end{bmatrix}^T|| \\ -||\begin{bmatrix}-0.5 & 0.5\end{bmatrix}^T - \begin{bmatrix}2 & 2\end{bmatrix}^T|| \end{bmatrix}\right)$$

$$= compet\left(\begin{bmatrix} -||\begin{bmatrix}-0.5 & -2\end{bmatrix}^T|| \\ -||\begin{bmatrix}-2.5 & -1.5\end{bmatrix}^T|| \end{bmatrix}\right) = compet\left(\begin{bmatrix}-2.06155 \\ -2.91548\end{bmatrix}\right) = \begin{bmatrix}1 \\ 0\end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_3 - w_{1_{old}}) = \begin{bmatrix}1.5 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}2 \\ 2\end{bmatrix} - \begin{bmatrix}1.5 \\ 0\end{bmatrix}) = \begin{bmatrix}1.5 \\ 0\end{bmatrix} + 0.5(\begin{bmatrix}0.5 \\ 2\end{bmatrix}) = \begin{bmatrix}1.5 \\ 0\end{bmatrix} + \begin{bmatrix}0.25 \\ 1\end{bmatrix}$$

$$w_{1_{new}} = \begin{bmatrix}1.75 \\ 1\end{bmatrix}$$

p_2:

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_2|| \\ -||w_2 - p_2|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -||\begin{bmatrix}1.75 & 1\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T|| \\ -||\begin{bmatrix}-0.5 & 0.5\end{bmatrix}^T - \begin{bmatrix}0 & 1\end{bmatrix}^T|| \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -||\begin{bmatrix}1.75 & 0\end{bmatrix}^T|| \\ -||\begin{bmatrix}-0.5 & -0.5\end{bmatrix}^T|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -1.75 \\ -0.707107 \end{bmatrix}\Big) = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

Κερδίζει ο δεύτερος νευρώνας άρα αλλάζουμε το w_2:

$$w_{2_{new}} = w_{2_{old}} + \alpha(p_2 - w_{2_{old}}) = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} + 0.5(\begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix}) = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} + 0.5(\begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}) = \begin{bmatrix} -0.5 \\ 0.5 \end{bmatrix} + \begin{bmatrix} 0.25 \\ 0.25 \end{bmatrix}$$
$$w_{2_{new}} = \begin{bmatrix} -0.25 \\ 0.75 \end{bmatrix}$$

p_3:

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_3|| \\ -||w_2 - p_3|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -||\begin{bmatrix}1.75 & 1\end{bmatrix}^T - \begin{bmatrix}2 & 2\end{bmatrix}^T|| \\ -||\begin{bmatrix}-0.25 & 0.75\end{bmatrix}^T - \begin{bmatrix}2 & 2\end{bmatrix}^T|| \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -||\begin{bmatrix}-0.25 & -1\end{bmatrix}^T|| \\ -||\begin{bmatrix}-2.25 & -1.25\end{bmatrix}^T|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -1.03078 \\ -2.57391 \end{bmatrix}\Big) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_3 - w_{1_{old}}) = \begin{bmatrix} 1.75 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} 2 \\ 2 \end{bmatrix} - \begin{bmatrix} 1.75 \\ 1 \end{bmatrix}) = \begin{bmatrix} 1.75 \\ 1 \end{bmatrix} + 0.5(\begin{bmatrix} 0.25 \\ 1 \end{bmatrix}) = \begin{bmatrix} 1.75 \\ 1 \end{bmatrix} + \begin{bmatrix} 0.125 \\ 0.5 \end{bmatrix}$$
$$w_{1_{new}} = \begin{bmatrix} 1.875 \\ 1.5 \end{bmatrix}$$

p_1:

$$a^1 = compet(n^1) = compet\Big(\begin{bmatrix} -||w_1 - p_1|| \\ -||w_2 - p_1|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -||\begin{bmatrix}1.875 & 1.5\end{bmatrix}^T - \begin{bmatrix}2 & 0\end{bmatrix}^T|| \\ -||\begin{bmatrix}-0.25 & 0.75\end{bmatrix}^T - \begin{bmatrix}2 & 0\end{bmatrix}^T|| \end{bmatrix}\Big)$$
$$= compet\Big(\begin{bmatrix} -||\begin{bmatrix}-0.125 & 1.5\end{bmatrix}^T|| \\ -||\begin{bmatrix}-2.25 & 0.75\end{bmatrix}^T|| \end{bmatrix}\Big) = compet\Big(\begin{bmatrix} -1.5052 \\ -2.37171 \end{bmatrix}\Big) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Κερδίζει ο πρώτος νευρώνας άρα αλλάζουμε το w_1:

$$w_{1_{new}} = w_{1_{old}} + \alpha(p_1 - w_{1_{old}}) = \begin{bmatrix}1.875\\1.5\end{bmatrix} + 0.5(\begin{bmatrix}2\\0\end{bmatrix} - \begin{bmatrix}1.875\\1.5\end{bmatrix}) = \begin{bmatrix}1.875\\1.5\end{bmatrix} + 0.5(\begin{bmatrix}0.125\\-1.5\end{bmatrix})$$
$$= \begin{bmatrix}1.875\\1.5\end{bmatrix} + \begin{bmatrix}0.0625\\-0.75\end{bmatrix} w_{1_{new}} = \begin{bmatrix}1.9375\\0.75\end{bmatrix}$$

Συνεπώς τα τελικά βάρη είναι:

$$w_1 = \begin{bmatrix}1.9375\\0.75\end{bmatrix}, w_2 = \begin{bmatrix}-0.25\\0.75\end{bmatrix}$$

Παρακάτω παραθέτετουμε τον κώδικα για το πρόβλημα:

```python
import math
import matplotlib.pyplot as plt
import numpy as np
def competitive_layer(input_list):
    """
    Competitive layer that returns the index of the biggest number,
    with the smallest index breaking ties.
    """
    max_index = 0
    max_value = input_list[0]

    for i in range(1, len(input_list)):
        if input_list[i] > max_value:
            max_index = i
            max_value = input_list[i]
        elif input_list[i] == max_value:
            max_index = min(max_index, i)

    result = [0] * len(input_list)
    result[max_index] = 1

    return result

# Kohonen Rule for update
def update_weights(a1, W, learning_rate, p):
    index = a1.index(1)

    difference = [p[i] - W[index][i] for i in range(len(p))]
    product = [learning_rate * element for element in difference]
```

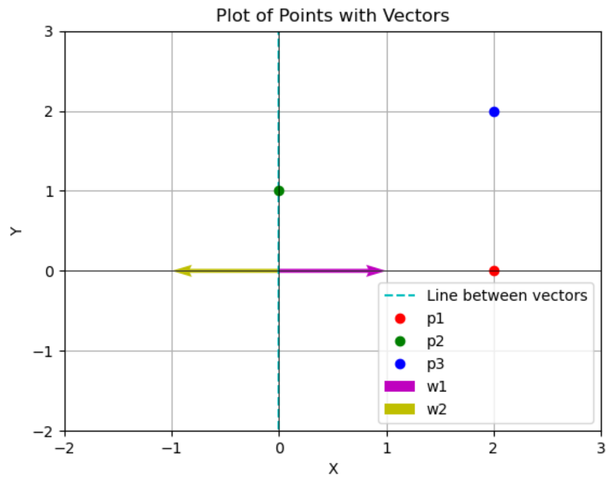
```
    W[index] = [W[index][i] + product[i] for i in range(len(product))]

    return W

def iteration(p, W, learning_rate):
    n1 = []
    for i in range(0, len(W)):
        n1.append(-math.sqrt((W[i][0] - p[0])**2 + (W[i][1] - p[1])**2))

    a1  = competitive_layer(n1)
    W = update_weights(a1, W, learning_rate, p)

    return W

def plot(p1, p2, p3, W):

    # Extract x and y coordinates
    p1_x, p1_y = p1
    p2_x, p2_y = p2
    p3_x, p3_y = p3
    w1_x, w1_y = W[0]
    w2_x, w2_y = W[1]

    vector1 = np.array(W[0])
    vector2 = np.array(W[1])

    # Calculate the angles of the vectors in radians
    angle_radians_vector1 = np.arctan2(vector1[1], vector1[0])
    angle_radians_vector2 = np.arctan2(vector2[1], vector2[0])

    # Calculate the angle difference between the two vectors
    angle_difference = (angle_radians_vector2 - angle_radians_vector1) /2
        + angle_radians_vector1

    # Plot the line with the angle difference from the origin (0,0)
    x_line = np.linspace(-1, 1, 100)
    y_line = np.tan(angle_difference) * x_line
    plt.plot(x_line, y_line, label=f'Line between vectors', linestyle='--', color='c')

    # Plot points
    plt.plot(p1_x, p1_y, 'ro', label='p1')
```

```
    plt.plot(p2_x, p2_y, 'go', label='p2')
    plt.plot(p3_x, p3_y, 'bo', label='p3')

    plt.quiver(0, 0, w1_x, w1_y, angles='xy', scale_units='xy',
        scale=1, color='m', label='w1')
    plt.quiver(0, 0, w2_x, w2_y, angles='xy', scale_units='xy',
        scale=1, color='y', label='w2')

    # Add labels and legend
    plt.xlabel('X')
    plt.ylabel('Y')
    plt.title('Plot of Points with Vectors')
    plt.legend()

    # Set axis limits
    plt.xlim(-2, 3)
    plt.ylim(-2, 3)

    # Show plot
    plt.grid(True)
    plt.axhline(0, color='black',linewidth=0.5)
    plt.axvline(0, color='black',linewidth=0.5)
    plt.show()

if __name__ == "__main__":

    # Init points and weights

    p1 = [2, 0]
    p2 = [0, 1]
    p3 = [2, 2]

    w1 = [1, 0]
    w2 = [-1, 0]

    W = [w1, w2]

    # Learning rate
    learning_rate = 0.5

    training_set1 = [p1, p2, p3, p2, p3, p1]
```

```
training_set2 = [p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3,
p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3,
p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1,
p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3,
p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1, p1, p2, p3,
p2, p3, p1,p1, p2, p3, p2, p3, p1, p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1,
p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3, p1,p1, p2, p3, p2, p3,
p1]
print(W)
plot(p1, p2, p3, W)

# In order to make the full training swape training_set_1 with training_set_2
#and remove the plot()-print() from the for loop in order to escape plotting every
updated weights
# Just put the plot()-print() outside the for loop to show the final state.
for p in training_set1:
    W = iteration(p, W, learning_rate)
    # print(W)
    # plot(p1, p2, p3, W)


print("Trained weights are: ")
print(W)
plot(p1, p2, p3, W)
```

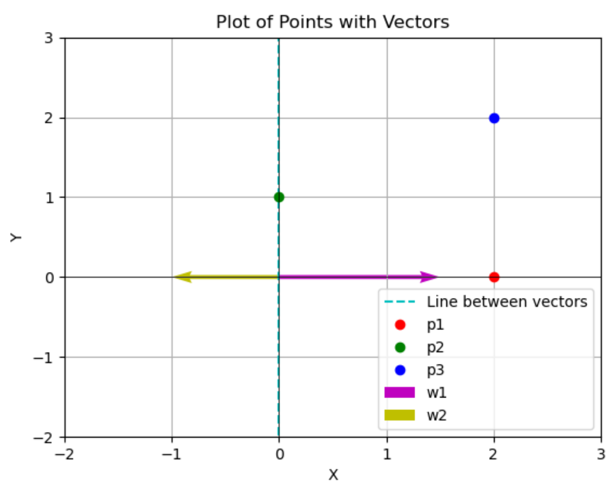
output:
Trained weights are:
$[[1.9375, 0.75], [-0.25, 0.75]]$

Το αποτέλεσμα μας με το χέρι συμπίπτει με το αποτέλεσμα που υπολογίσαμε και με τον κώδικα. Επίσης βλέπουμε και σχηματικά πως αλλάζουν οι τιμές των βαρών σε κάθε επανάληψη, ενώ έχουμε εκτυπώσει και την γραμμή που χωρίζει τις δύο κλάσεις.

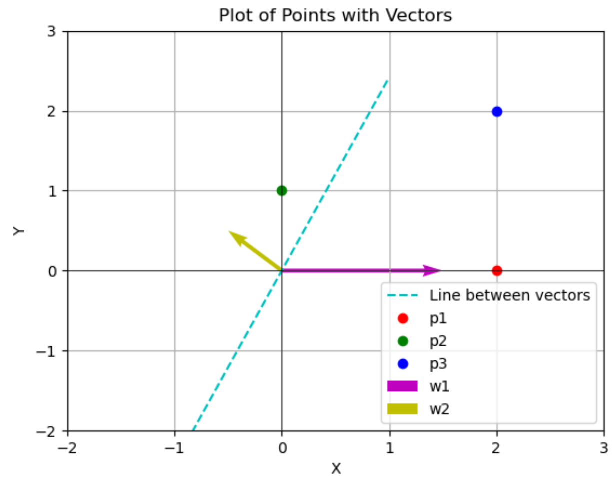


Σχήμα 13. w1=[0, 1], w2[-1, 0] (Αρχική κατάσταση)

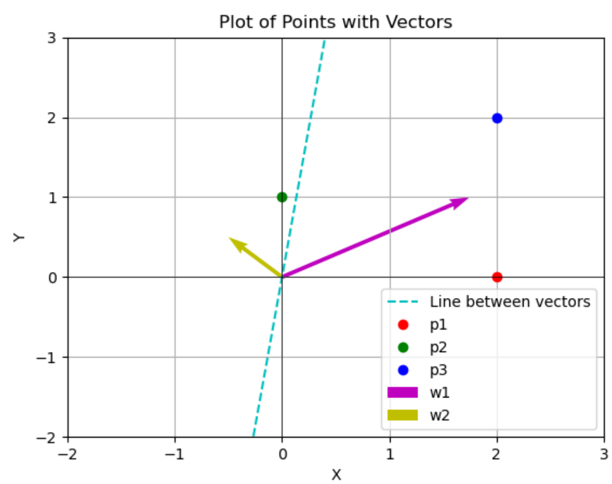
[H]



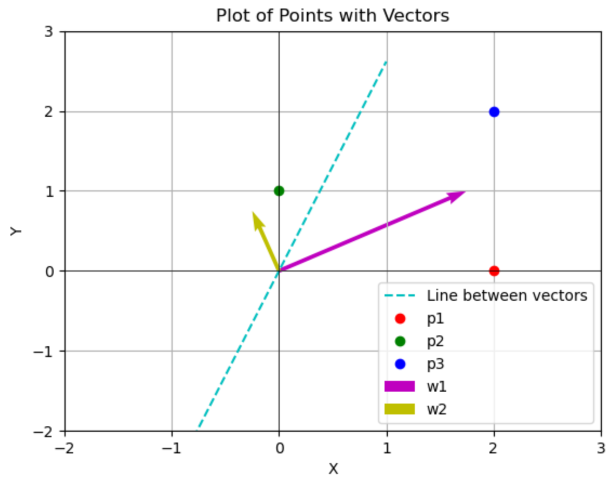
Σχήμα 14. w1=[1.5, 0], w2[-1, 0]



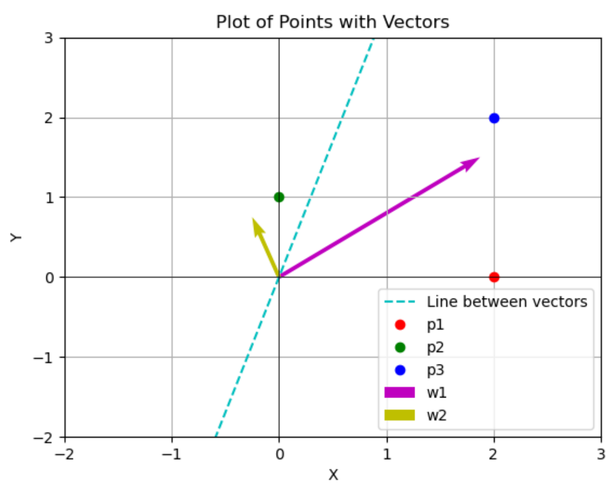
Σχήμα 15. w1=[1.5, 0], w2[-0.5, 0.5]



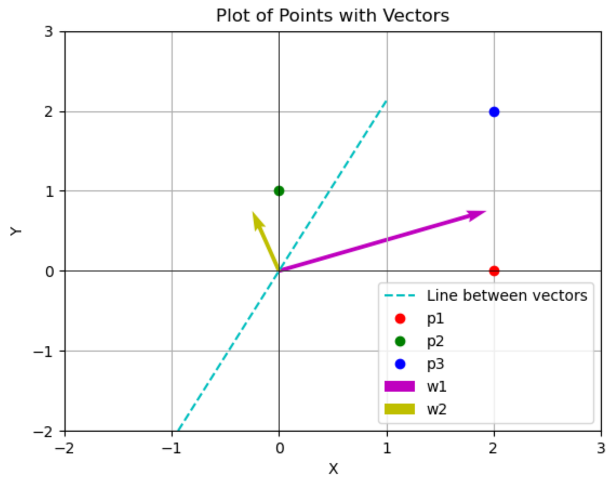
Σχήμα 16. w1=[1.75, 1], w2[-0.5, 0.5]



Σχήμα 17. w1=[1.75, 1], w2[-0.25, 0.75]

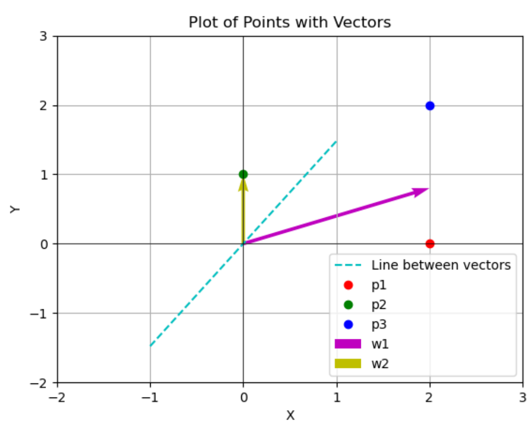


Σχήμα 18. w1=[1.875, 1.5], w2[-0.25, 0.75]



Σχήμα 19. w1=[1.9375, 0.75], w2[-0.25, 0.75] (Τελική κατάσταση)

Τέλος επαναλάβαμε την διαδικασία του script χρησιμοποιώντας την αρχική ακολουθία εισόδου πολλές φορές (training_set_2) και αυτό είναι το τελικό αποτέλεσμα. Παρατηρούμε ότι το w_1 'πέφτει στη μέση απο τα σημεία p_1 και p_3, ενώ το w_2 είναι σχεδόν ακριβώς πάνω στο p_2. Συνεπώς στην κλάση 1 ανήκει p_1 και p_3, ενώ στην κλάση 2 ανήκει το p_2. Η μπλε γραμμή ορίζει τα όρια των δύο κλάσεων



Σχήμα 20. w1=[2, 0.8], w2[-0.008e-19, 0.999]

5 Problem-05

Αρχικά δημιουργούμε τα samples από το autoregressive model, έπειτα δημιουργούμε ένα dataset χρησιμοποιώντας sequence len = 5. Έπειτα φτιάχνουμε μια κλάση για το recuurent neural network LSTM τα βήματα του training για κάθε epoch είναι το forward και το backward για update των παραμέτρων του LSTM

Ύστερα από δοκιμές καταλήξαμε στα εξής hyperparameters:

- learning_rate = 0.0001
- nepoch = 20
- T = sequence_length
- hidden_dim = hidden_dim
- output_dim = 1
- bptt_truncate = 5
- min_clip_value = -5
- max_clip_value = 5

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

def generate_AR_samples(num_samples, seed):
    # np.random.seed(seed)
    a1, a2, a3 = 0.5, -0.1, 0.2

    samples = np.zeros(num_samples)
    samples[:3] = np.random.rand(1, 3)
    for i in range(3, num_samples):
        samples[i] = a1 * samples[i-1] + a2 * samples[i-2] + a3 * samples[i-3]
                + np.random.uniform(-0.25, 0.25)

    return samples

def create_dataset(samples, sequence_length):
    dataX, dataY = [], []
    for i in range(len(samples) - sequence_length):
        seq_in = samples[i:i + sequence_length]
        seq_out = samples[i + sequence_length]
        dataX.append(seq_in)
        dataY.append(seq_out)
```

```
    X = np.array(dataX)
    Y = np.array(dataY)

    return np.expand_dims(X, axis=2), np.expand_dims(Y, axis=1)

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

class LSTM():
    def __init__(self, sequence_length, hidden_dim):
        self.learning_rate = 0.0001
        self.nepoch = 20
        self.T = sequence_length              # length of sequence
        self.hidden_dim = hidden_dim
        self.output_dim = 1
        self.bptt_truncate = 5
        self.min_clip_value = -5
        self.max_clip_value = 5
        self.U = np.random.uniform(0, 1, (self.hidden_dim, self.T))
        self.W = np.random.uniform(0, 1, (self.hidden_dim, self.hidden_dim))
        self.V = np.random.uniform(0, 1, (self.output_dim, self.hidden_dim))
    def train(self, ):

        for epoch in range(self.nepoch):

            # train model
            for i in range(Y.shape[0]):
                x, y = X[i], Y[i]

                layers = []
                prev_s = np.zeros((hidden_dim, 1))
                dU = np.zeros(self.U.shape)
                dV = np.zeros(self.V.shape)
                dW = np.zeros(self.W.shape)

                dU_t = np.zeros(self.U.shape)
                dV_t = np.zeros(self.V.shape)
                dW_t = np.zeros(self.W.shape)

                dU_i = np.zeros(self.U.shape)
                dW_i = np.zeros(self.W.shape)
```

```
# forward pass
for t in range(self.T):
    new_input = np.zeros(x.shape)
    new_input[t] = x[t]
    mulu = np.dot(self.U, new_input)
    mulw = np.dot(self.W, prev_s)
    add = mulw + mulu
    s = sigmoid(add)
    mulv = np.dot(self.V, s)
    layers.append({'s':s, 'prev_s':prev_s})
    prev_s = s

        # derivative of pred
dmulv = (mulv - y)

# backward pass
for t in range(self.T):
    dV_t = np.dot(dmulv, np.transpose(layers[t]['s']))
    dsv = np.dot(np.transpose(self.V), dmulv)

    ds = dsv
    dadd = add * (1 - add) * ds

    dmulw = dadd * np.ones_like(mulw)

    dprev_s = np.dot(np.transpose(self.W), dmulw)


    for i in range(t-1, max(-1, t-self.bptt_truncate-1), -1):
        ds = dsv + dprev_s
        dadd = add * (1 - add) * ds

        dmulw = dadd * np.ones_like(mulw)
        dmulu = dadd * np.ones_like(mulu)

        dW_i = np.dot(self.W, layers[t]['prev_s'])
        dprev_s = np.dot(np.transpose(self.W), dmulw)

        new_input = np.zeros(x.shape)
        new_input[t] = x[t]
```

```
                dU_i = np.dot(self.U, new_input)
                dx = np.dot(np.transpose(self.U), dmulu)

                dU_t += dU_i
                dW_t += dW_i

            dV += dV_t
            dU += dU_t
            dW += dW_t

            if dU.max() > self.max_clip_value:
                dU[dU > self.max_clip_value] = self.max_clip_value
            if dV.max() > self.max_clip_value:
                dV[dV > self.max_clip_value] = self.max_clip_value
            if dW.max() > self.max_clip_value:
                dW[dW > self.max_clip_value] = self.max_clip_value


            if dU.min() < self.min_clip_value:
                dU[dU < self.min_clip_value] = self.min_clip_value
            if dV.min() < self.min_clip_value:
                dV[dV < self.min_clip_value] = self.min_clip_value
            if dW.min() < self.min_clip_value:
                dW[dW < self.min_clip_value] = self.min_clip_value

        # update
        self.U -= self.learning_rate * dU
        self.V -= self.learning_rate * dV
        self.W -= self.learning_rate * dW
def eval(self, X, Y):
    preds = []
    for i in range(Y.shape[0]):
        x, y = X[i], Y[i]
        prev_s = np.zeros((self.hidden_dim, 1))
        # Forward pass
        for t in range(self.T):
            mulu = np.dot(self.U, x)
            mulw = np.dot(self.W, prev_s)
            add = mulw + mulu
            s = sigmoid(add)
            mulv = np.dot(self.V, s)
```

```
            prev_s = s

        preds.append(mulv)
    return np.array(preds)

if __name__ == "__main__":

    num_samples = 200
    ar_samples = generate_AR_samples(num_samples, seed=32)
    val_ar_samples = generate_AR_samples(num_samples, seed=23)

    sequence_length = 5
    hidden_dim = 100
    X, Y = create_dataset(ar_samples, sequence_length)

    X_val, Y_val = create_dataset(val_ar_samples, sequence_length)

    lstm  = LSTM(sequence_length, hidden_dim)

    lstm.train()

    preds = lstm.eval(X_val, Y_val)

    plt.plot(preds[:, 0, 0], 'g', label="predict")
    plt.plot(Y_val[:, 0], 'r', label="actual")
    plt.legend()
    plt.show()

    mse = np.mean((preds-Y_val)**2)
    print("MSE is: ")
    print(mse)
```

Σχήμα 21. Responses for validation dataset

output:
MSE is: 0.035892314932

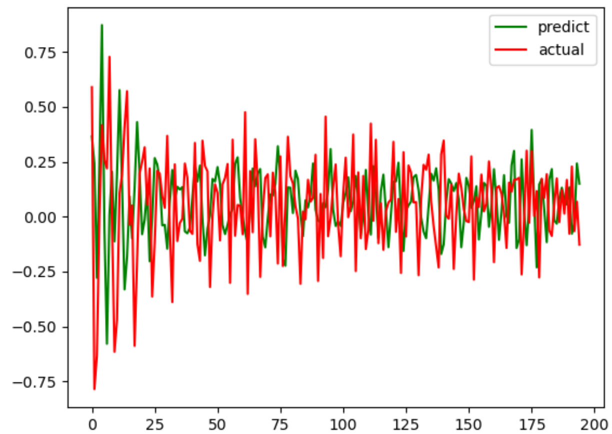
6 Problem-06

Η μόνη αλλαγή στον κώδικα μας είναι στην συνάρτηση που κάνει generate τα sample για τον Moving Average, ο υπόλοιπος κώδικας είναι ο ίδιος με το Problem 6.

```
def generate_MA_samples(num_samples, seed):
    # np.random.seed(seed)
    a1 = a2 = a3 = a4 = a5 = a6 = -0.5

    samples = np.zeros(num_samples)
    samples[:6] = np.random.rand(1, 6)
    for i in range(6, num_samples):
        samples[i] = a1 * samples[i-1] + a2 * samples[i-2] + a3 * samples[i-3]
            + a4 * samples[i-4] + a5 * samples[i-5] + a6 * samples[i-6]
                + np.random.uniform(0, 0.5)

    return samples
```

output:
MSE is:

0.08871415148662432



Σχήμα 22. Responses for validation dataset

7 Problem-07

- A. Large integers.
- B. Very small numbers.
- C. Medium-weight men.
- D. Numbers approximately between 10 and 20.

$\tilde{A}$ = ”Large integers” Αν θεωρήσουμε μεγάλους ακεραίους αυτούς μεγαλύτερους από το 1000

$$\tilde{A} = \{(x, \mu_{\tilde{A}}(x)) \mid x \in X\}$$

όπου

$$\mu_{\tilde{A}}(x) = \begin{cases} (1 + (x - 1000)^{-2})^{-1} & x \geq 1000 \\ 0 & x < 1000 \end{cases}$$

Θέλουμε μικρούς αριθμούς κοντά στο 0 άρα έχουμε:

$$\tilde{B} = \{(x, \mu_{\tilde{B}}(x)) \mid \mu(x) = (1 + (x)^2)^{-1}\}$$

$\tilde{C}$ = ”Medium-weight men” :
Αν θεωρήσουμε ότι οι medium weight men είναι αυτοί κοντά στα 80 κιλά

$$\tilde{C} = \{(x, \mu_{\tilde{C}}(x)) \mid \mu_{\tilde{C}}(x) = (1 + (x - 80)^2)^{-1}\}$$

$\tilde{D}$ = "Numbers approximately between 10 and 20"
Θα δηλώσουμε δυο fuzzy sets τα:

$$\tilde{D_1} = \text{"Numbers larger than 10"}$$

$$\tilde{D_2} = \text{"Numbers smaller than 20"}$$

$$\tilde{D_1} = \{(x, \mu_{\tilde{D_1}}(x)) \mid x \in X\}$$

όπου

$$\mu_{\tilde{D_1}}(x) = \begin{cases} 0 & x < 10 \\ (1 + (x-10)^{-2})^{-1} & x \geq 10 \end{cases}$$

και

$$\tilde{D_2} = \{(x, \mu_{\tilde{D_2}}(x)) \mid x \in X\}$$

όπου

$$\mu_{\tilde{D_2}}(x) = \begin{cases} (1 + (x-20)^{-2})^{-1} & x \leq 20 \\ 0 & x > 20 \end{cases}$$

Αρα εχουμε

$$\mu_{\tilde{D}}(x) = \mu_{\tilde{D_1} \cap \tilde{D_2}}(x) = \begin{cases} 0 & x \leq 10, x \geq 20 \\ min[(1 + (x-10)^{-2})^{-1}, (1 + (x-20)^{-2})^{-1}] & 10 < x < 20 \end{cases}$$

8 Problem-08

Έχουμε το fuzzy set R, το ordinary subset για α = 0.3 ορίζεται ως:

$$R_{0.3} = \{(x, y \mid \mu_{\tilde{R}}(x, y)) \geq 0.3\}$$

$$1 - \frac{1}{1 + x^2 + y^2} \geq 0.3 \Leftrightarrow x^2 + y^2 \ \geq \frac{3}{7}$$

Όπως βλέπουμε και στο σχήμα 23, γεωμετρικά έχουμε όλα τα σημεία εξωτερικά ενός κύκλου με κέντρο το (0,0) και ακτίνα $\sqrt{\frac{3}{7}}$

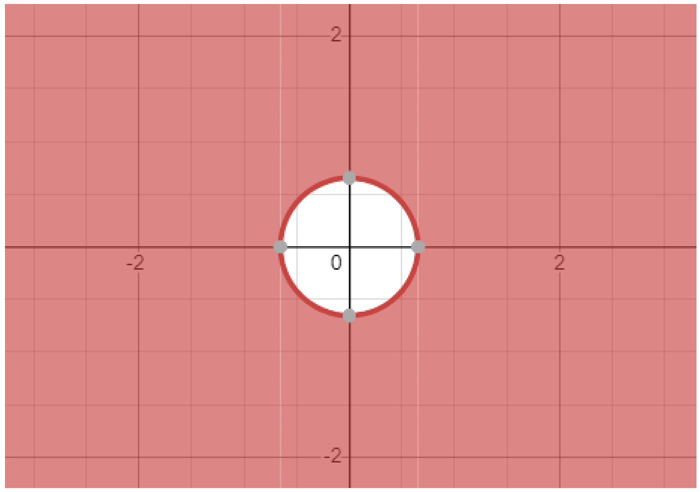
9 Problem-09

A.

Αρχικά βρίσκουμε το Generalized Hamming distance μεταξύ του $\tilde{A}, \bar{\tilde{A}}$

$$d(\tilde{A}, \bar{\tilde{A}}) = \int_0^n |\mu_{\tilde{A}}(x) - \mu_{\bar{\tilde{A}}}(x)| \ = \int_0^n |1 - 2\mu_{\tilde{A}}(x))|$$

$$= \int_0^a |1 - 2\frac{x^2}{\alpha^2}| = \int_0^{\frac{a}{\sqrt{2}}} 1 - 2\frac{x^2}{\alpha^2} - \int_{\frac{a}{\sqrt{2}}}^a 1 - 2\frac{x^2}{\alpha^2}$$



Σχήμα 23. Graph of the ordinary relation

$$= (x - \tfrac{2x^3}{3\alpha^2})\Big|_0^{\frac{a}{\sqrt{2}}} - (x - \tfrac{2x^3}{3\alpha^2})\Big|_{\frac{a}{\sqrt{2}}}^{a} = \frac{a}{\sqrt{2}} - \frac{2}{3\alpha^2}\frac{a^3}{2\sqrt{2}} - 0 - (a - \frac{2a^3}{3\alpha^2} - \frac{a}{\sqrt{2}} + \frac{2}{3\alpha^2}\frac{a^3}{2\sqrt{2}})$$

$$= \frac{\sqrt{2}a}{3} - (\frac{a}{3} - \frac{\sqrt{2}a}{3}) = (\frac{2\sqrt{2}}{3} - \frac{1}{3})a$$

Υπολογίζουμε το linear index of fuzziness:

$$v(\tilde{A}) = \frac{2}{a}d(\tilde{A}, \bar{\tilde{A}}) = \frac{2}{a}\frac{2\sqrt{2}-1}{3}a = \frac{4\sqrt{2}-2}{3} \approx 1.219$$

B.

Δουλεύουμε στο διάστημα [0, α/2]

$$d(\tilde{A}, \bar{\tilde{A}}) = \int_0^{\frac{a}{2}} |1 - 2\frac{4x^2}{\alpha^2}| = \int_0^{\frac{a}{2}} |1 - 2\frac{x^2}{\frac{\alpha}{2}^2}| = (\frac{2\sqrt{2}}{3} - \frac{1}{3})\frac{a}{2}$$

Με αντικατάσταση του α/2 προκύπτει το ολοκλήρωμα του ερωτήματος Α. Το ολοκλήρωμα στο διάστημα [α/2, α] ισούται με αυτό στο [0, α/2] καθώς είναι η μετατόπισή της συνάρτησης κατά α δεξιά και η συνάρτηση είναι συμμετρική άρα συνολικά στο διάστημα [0,α] έχουμε :

$$d(\tilde{A}, \bar{\tilde{A}}) = (\frac{2\sqrt{2}}{3} - \frac{1}{3})a$$

και ομοίως με τω ερώτημα Α.

$$v(\tilde{A}) = \frac{2}{a}d(\tilde{A}, \bar{\tilde{A}}) = \frac{2}{a}\frac{2\sqrt{2}-1}{3}a = \frac{4\sqrt{2}-2}{3} \approx 1.219$$

10 Problem-10

Θελουμε το max-min composition των δυο fuzzy relations R_1, R_2:

$$\mu_{\tilde{R_1}\circ\tilde{R_2}}(x,z) = \max_y \left[\min\{\mu_{mR_1(x,y)}, \mu_{mR_2(y,z)}\}\right] = \max_y \left[\min\{e^{-k(x-y)^2}, e^{-k(y-z)^2}\}\right]$$

Έστω ένα σημείο x_1, z_1 της σύνθεσής στο γράφημά χρησιμοποιούμε k=1 για μεγαλύτερά k άπλα στενεύει η παράβολη. Το max από τα ελάχιστα μεταξύ των δυο σημείων βρίσκεται στο σημείο τομής των δυο συναρτήσεων :

$$e^{-k(x_1-y)^2} = e^{-k(y-z_1)^2} \Leftrightarrow -k(x_1-y)^2 = -k(y-z_1)^2 \Leftrightarrow$$

$$x_1^2 - 2x_1y + y^2 = y^2 - 2yz_1 + z_1^2 \Leftrightarrow 2y(z_1 - x_1) = z_1^2 - x_1^2$$

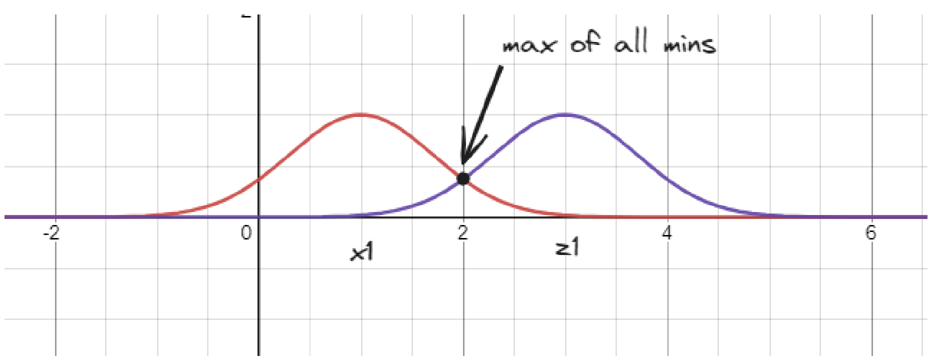
$$\Leftrightarrow y = \frac{(z_1 - x_1)(z_1 + x_1)}{2(z_1 - x_1)} \Leftrightarrow y = \frac{z_1 + x_1}{2} \quad , x_1 \neq z_1$$

Άρα για $x_1 \neq z_1$ το compostition παίρνει τιμή $e^{-k(x_1-(\frac{z_1+x_1}{2})^2} = e^{-k(\frac{x_1-z_1}{2})^2}$
Για την περίπτωση $x_1 = z_1$ η συναρτήσεις ταυτίζονται οπότε το μέγιστο ελάχιστο είναι το μέγιστο της συνάρτησης που βρίσκεται στο σημείο

$$y = x_1 = z_1 = \frac{x_1 + z_1}{2}$$

Άρα για όλα τα σημεία x_1, z_1 έχουμε την τιμή της συνάρτησής δηλαδή :

$$\mu_{\tilde{R_1}\circ\tilde{R_2}}(x,z) = e^{-k(\dfrac{x-z}{2})^2}, \quad k \geq 1$$



Σχήμα 24. Fuzzy reations for $x_1 = 1, \quad z_1 = 3$