Τσαντίκης Γεώργιος
ΑΕΜ: 10722

Αναφορά 3ης υποχρεωτικής εργασίας στο μάθημα «ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ- ΒΑΘΙΑ ΜΑΘΗΣΗ»

ΠΕΡΙΕΧΟΜΕΝΑ:

1. Εισαγωγή

Στην παρούσα εργασία πραγματοποιήθηκε η υλοποίηση από την αρχή (**from scratch**), με κώδικα **Python**, ενός **Radial Basis Function Neural Network** (RBFNN) το οποίο εκπαιδεύτηκε για τον διαχωρισμό όλων των κλάσεων που υπάρχουν στη βάση δεδομένων **CIFAR-10**. Η CIFAR-10 περιέχει 60.000 έγχρωμες (RGB) εικόνες μεγέθους 32x32 pixels, κατανεμημένες σε 10 διαφορετικές κατηγορίες /κλάσεις (π.χ., σκύλος, φορτηγό, αεροπλάνο), από τις οποίες 50.000 εικόνες είναι δείγματα εκπαίδευσης και 10.000 δείγματα ελέγχου.

Η εργασία επικεντρώνεται στην υλοποίηση και μελέτη ενός **RBFNN**, χωρίς τη χρήση έτοιμων συναρτήσεων από βιβλιοθήκες μηχανικής μάθησης (όπως Scikit-Learn ή TensorFlow), με στόχο την κατανόηση της λειτουργίας των επιμέρους αλγορίθμων και μηχανισμών που συμμετέχουν στη διαδικασία εκπαίδευσης και πρόβλεψης. Χρησιμοποιήθηκαν τεχνικές όπως:

- **Κανονικοποίηση** εισόδων ώστε οι τιμές των pixel να βρίσκονται στο **εύρος [0, 1]** και **One-Hot Encoding** για τα τις κλάσεις.
- Εφαρμογή **PCA (Principal Component Analysis)** για τη μείωση διάστασης των δεδομένων και την αποδοτικότερη εκπαίδευση του RBFNN, .
- Επιλογή **κέντρων** των συναρτήσεων RBF όχι μόνο με τη χρήση του κανόνα **K-Μέσων** (k-means) αλλά και **τυχαία**.

Για την εκπαίδευση του RBFNN:

- **Κανόνας Δέλτα** (ADALINE) με:
    - Χρήση **Cross-Entropy Loss** για τον υπολογισμό της απώλειας, σε συνδυασμό με **Softmax** στο εξωτερικό επίπεδο.
    - Ρύθμιση **παραμέτρων** εκπαίδευσης όπως learning rate, epochs, αριθμός νευρώνων και sigma ($beta = \frac{1}{2 \cdot sigma^2}$)
- **Least Squares** Method με **regularization**

Στόχοι της εργασίας είναι :

- Η διερεύνηση της επίδρασης διαφορετικών τιμών **παραμέτρων** στην απόδοση του RBFNN.
- Η παρουσίαση χαρακτηριστικών **παραδειγμάτων** ορθής και εσφαλμένης **κατηγοριοποίησης** από το μοντέλο.
- Η **σύγκριση** της απόδοσης του **RBFNN** με άλλες μεθόδους κατηγοριοποίησης, όπως **Nearest Neighbor** (1-NN, 3-NN) και **Nearest Class Centroid** (NCC).

2. Περιγραφή αλγορίθμων – Ανάλυση Κώδικα

2.1. Βιβλιοθήκες

Οι παρακάτω βιβλιοθήκες αποτελούν απαραίτητα εργαλεία για την κάλυψη των απαιτήσεων της εργασίας :

- **import numpy as np** → για την αποδοτική διαχείριση αριθμητικών υπολογισμών και πολυδιάστατων πινάκων (x_train, x_test κτλ.)
- **import time** → για την μέτρηση του χρόνου εκπαίδευσης
- **import pickle** → για την αποθήκευση και ανάκτηση των δεδομένων σε δυαδική μορφή
- **import requests** → για την αποστολή αιτήματος HTTP, με σκοπό την λήψη του dataset CIFAR-10 από το διαδίκτυο
- **import tarfile** → για την αποσυμπίεση των δεδομένων (εξάγει αρχεία .tar.gz)
- **import os**→ για τον χειρισμό λειτουργιών του συστήματος αρχείων (έλεγχος ύπαρξης αρχείων κτλ.)

2.2. Download και extract του dataset CIFAR-10

Η παρακάτω συνάρτηση εξυπηρετεί τη λήψη από το διαδίκτυο και την αποσυμπίεση του dataset σε περίπτωση που το συμπιεσμένο αρχείο με όνομα *"cifar-10-python.tar.gz"* δεν υπάρχει στον τοπικό φάκελο [*"os.path.exists()"*] στον οποίο εξάγει το περιεχόμενο του σε δυαδική μορφή. Πλέον, τα δεδομένα του CIFAR-10 είναι έτοιμα για περαιτέρω επεξεργασία σε μη συμπιεσμένη μορφή στον φάκελο "cifar-10-batches-py".

```
# Download και extract του dataset CIFAR-10
def download_and_extract_cifar10():
    url = "https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz"
    archive_name = "cifar-10-python.tar.gz"
    if not os.path.exists(archive_name):
        print("Downloading CIFAR-10 dataset...")
        with requests.get(url, stream=True) as r:
            with open(archive_name, "wb") as file: # ανοίγει για γράψιμο
                file.write(r.content)
    if not os.path.exists("cifar-10-batches-py"):
        print("Extracting dataset...")
        with tarfile.open(archive_name, "r:gz") as tar:
            tar.extractall()
```

2.3. Φόρτωση και προεπεξεργασία του dataset

Στη συνάρτηση *"load_cifar10() "* περιέχεται η βοηθητική συνάρτηση *"unpickle(file)"* η οποία διαβάζει τα δεδομένα από αρχεία που είναι αποθηκευμένα σε δυαδική μορφή και χρησιμοποιείται για τη φόρτωση (*pickle*) των δεδομένων του CIFAR-10 ύστερα από την λήψη και αποθήκευση τους στο φάκελο *"./cifar-10-batches-py"* με χρήση της συνάρτησης *" download_and_extract_cifar10() "* της παραγράφου *2.2, σελ.3.* Τα δεδομένα εκπαίδευσης περιλαμβάνονται σε πέντε batches (αρχεία data_batch_1 έως data_batch_5), με κάθε **batch** να περιέχει τις εικόνες και τις αντίστοιχες ετικέτες τους, δηλαδή τις κατηγορίες/κλάσεις στις οποίες ανήκουν. Οι εικόνες φορτώνονται και αποθηκεύονται στον πίνακα ***x*** και οι ετικέτες στον πίνακα ***y***.

Έπειτα πραγματοποιείται η **κανονικοποίηση**, που μετατρέπει τις τιμές των εικόνων από το εύρος [0, 255] στο εύρος [0, 1], διαιρώντας με το μέγιστο (255.0), μειώνοντας τη διακύμανση μεταξύ των εισόδων. Έτσι διευκολύνεται η σύγκλιση κατά την εκπαίδευση, καθώς οι βελτιώσεις στα βάρη γίνονται πιο ομαλά**.** Επιπλέον, με αυτόν τον τρόπο αποφεύγουμε το overflow error στο exponential της **SoftMax** (βλ. *2.8, σελ.10*). Αντίστοιχη επεξεργασία υφίστανται τα δεδομένα ελέγχου που περιέχονται στο αρχείο *"test_batch"*. Τελικά, τα κανονικοποιημένα δεδομένα αποθηκεύονται ως **NumPy Arrays** στις μεταβλητές *"x_train"* (δείγματα εκπαίδευσης) και *"x_test"* (δείγματα ελέγχου). Τελικά, η συνάρτηση επιστρέφει τις μεταβλητές *"x_train"* , *"y_train"* (ετικέτες των x_train), *"x_test"*, *"y_test"* (ετικέτες των x_test).

```python
# Φόρτωση των δεδομένων του dataset
def load_cifar10():
    def unpickle(file):
        with open(file, 'rb') as fo:
            data = pickle.load(fo, encoding='bytes')
        return data

    download_and_extract_cifar10()
    data_path = "./cifar-10-batches-py/"
    x, y = [], []

    for i in range(1, 6):
        batch = unpickle(os.path.join(data_path, f"data_batch_{i}"))
        x.append(batch[b'data'])
        y.extend(batch[b'labels'])

    x_train = np.vstack(x).astype(np.float32) / 255.0 # Κανονικοποίηση τιμών
    y_train = np.array(y)

    test_batch = unpickle(os.path.join(data_path, "test_batch"))
    x_test = test_batch[b'data'].astype(np.float32) / 255.0
    y_test = np.array(test_batch[b'labels'])

    return x_train, y_train, x_test, y_test
```

2.4. PCA – Μείωση διάστασης δεδομένων

Η συνάρτηση *"pca(X, n_components=0.91)"* εφαρμόζει την τεχνική **Ανάλυσης Κύριων Συνιστωσών (PCA)** στο dataset και στηρίζεται στα ιδιοδιανύσματα του πίνακα συνδιασποράς που έχουμε να χρησιμοποιήσουμε. Έχει σκοπό την συμπίεση των δεδομένων, διατηρώντας παράλληλα τη μέγιστη δυνατή πληροφορία. Θέλει δηλαδή να βρει έναν υποχώρο διάστασης $d < p$ (p η διάσταση του χώρου δεδομένων) που εξηγεί όσο το δυνατόν περισσότερο τη διασπορά των σημείων. Η συνάρτηση δέχεται ως είσοδο:

- *"X" : τα δεδομένα που πρόκειται να υποβληθούν σε ανάλυση*
- *"n_components": Το ποσοστό της διατηρούμενης πληροφορίας (variance), με προεπιλεγμένη τιμή 0.91 (91% της διακύμανσης)*

```
# PCA implementation
def pca(X, n_components=0.91):
    X_mean = np.mean(X, axis=0)
    X_centered = X - X_mean # κεντράρισμα των τιμών

    # Υπολογισμός πίνακα συνδιασποράς, ιδιοτιμών και ιδιοδιανυσμάτων
    # > rowvar=false: οι στήλες του πίνακα αντιπροσωπεύουν τα χαρακτηριστικά
    cov_matrix = np.cov(X_centered, rowvar=False)
    eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)

    # Ταξινόμηση των ιδ/μων για επιλογή των σημαντικότερων ιδ/μών και ιδ/των
    # με [::-1] ώστε η σειρά να είναι φθίνουσα
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    # Υπολογισμός της συνολικής διακύμανσης, της σωρευτικής εξηγούμενης
    # διακύμανσης και των απαιτούμενων κύριων συνιστωσών
    total_variance = np.sum(eigenvalues)
    explained_variance = np.cumsum(eigenvalues) / total_variance
    num_components = np.argmax(explained_variance >= n_components) + 1

    print(f"Επιλέχθηκαν {num_components} συνιστώσες για διατήρηση {n_components*100}% διακύμανσης.")

    # Προβολή στον νέο χώρο χαμηλότερης διάστασης
    selected_eigenvectors = eigenvectors[:, :num_components]
    reduced_X = np.dot(X_centered, selected_eigenvectors)

    return reduced_X, selected_eigenvectors, X_mean
```

Αναλυτικότερα:

I. Για τον υπολογισμό των **ιδιοδιανυσμάτων** (*"eigenvectors"*) του πίνακα συνδιασποράς (*"cov_matrix"*), απαραίτητη διαδικασία αποτελεί το **κεντράρισμα** των τιμών, αφαιρώντας τον **μέσο όρο** από κάθε χαρακτηριστικό (στήλη του X από pixels) του dataset, ώστε η διακύμανση να μετριέται γύρω από το μηδέν. Οι **ιδιοτιμές** (*"eigenvalues"*) αντιπροσωπεύουν την διακύμανση που εξηγεί κάθε ιδιοδιάνυσμα:

```
X_mean = np.mean(X, axis=0)
X_centered = X - X_mean # κεντράρισμα των τιμών

# Υπολογισμός πίνακα συνδιασποράς, ιδιοτιμών και ιδιοδιανυσμάτων
# - rowvar=false: οι στήλες του πίνακα αντιπροσωπεύουν τα χαρακτηριστικά
cov_matrix = np.cov(X_centered, rowvar=False)
eigenvalues, eigenvectors = np.linalg.eigh(cov_matrix)
```

II. Οι ιδιοτιμές **ταξινομούνται** σε φθίνουσα σειρά με βάση τους δείκτες τους("sorted_indices") και αντίστοιχα με αυτές ταξινομούνται και τα ιδιοδιανύσματα. Αυτό συμβαίνει, επειδή θέλουμε να διατηρήσουμε τις **πρώτες κύριες συνιστώσες**, που εξηγούν τη **μεγαλύτερη διακύμανση** στα δεδομένα και αντιστοιχούν στα **ιδιοδιανύσματα** που σχετίζονται με τις **μεγαλύτερες ιδιοτιμές**:

```
# Ταξινόμηση των ιδ/μων για επιλογή των σημαντικότερων ιδ/μων και ιδ/των
# με [::-1] ώστε η σειρά να είναι φθίνουσα
sorted_indices = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
```

III. Υπολογίζεται η **συνολική διακύμανση** των δεδομένων ("*total_variance*"), η **σωρευτική** (cumulative) εξηγούμενη διακύμανση ("*explained_variance*") για να δούμε πόση διακύμανση εξηγούν οι κύριες συνιστώσες και ο **αριθμός των κύριων συνιστωσών** ("*num_components*") που χρειάζονται ώστε να διατηρείται το καθορισμένο ποσοστό της συνολικής διακύμανσης:

```
# Υπολογισμός της συνολικής διακύμανσης, της σωρευτικής εξηγούμενης
# διακύμανσης και των απαιτούμενων κύριων συνιστωσών
total_variance = np.sum(eigenvalues)
explained_variance = np.cumsum(eigenvalues) / total_variance
num_components = np.argmax(explained_variance >= n_components) + 1
```

IV. Επιλέγονται τα πρώτα "*num_components*" ιδιοδιανύσματα που αντιστοιχούν στις πρώτες κύριες συνιστώσες, και τα δεδομένα προβάλλονται στον νέο χώρο χαμηλότερης διάστασης. Η συνάρτηση επιστρέφει:

- *"reduced_X": Τα δεδομένα στον νέο, μειωμένο χώρο διάστασης, μετά την εφαρμογή του PCA*
- *"selected_eigenvectors": Τα επιλεγμένα ιδιοδιανύσματα (οι κύριες συνιστώσες) που χρησιμοποιήθηκαν για τη μείωση της διάστασης*
- *"X_mean": Ο αρχικός μέσος όρος των δεδομένων, που χρησιμοποιείται για να επαναφέρει τα δεδομένα στην αρχική τους κλίμακα (αν χρειαστεί)*

```
# Προβολή στον νέο χώρο χαμηλότερης διάστασης
selected_eigenvectors = eigenvectors[:, :num_components]
reduced_X = np.dot(X_centered, selected_eigenvectors)

return reduced_X, selected_eigenvectors, X_mean
```

Οι μεταβλητές *"selected_eigenvectors" και "X_mean"* θα χρειαστούν στην **εφαρμογή PCA στα δείγματα ελέγχου.**

Η μείωση της διάστασης μέσω PCA βελτιώνει την εκπαίδευση του RBFNN, διότι απομακρύνει θόρυβο και περιττές πληροφορίες και μειώνει την πολυπλοκότητα των υπολογισμών.

*Οι συναρτήσεις που παρουσιάζονται στις παραγράφους 2.5 και 2.6 αφορούν το **κρυφό (RBF) επίπεδο**. Πιο συγκεκριμένα, υλοποιούν την εκπαίδευση του κρυφού στρώματος και τις εξόδους (ενεργοποίηση) των κρυφών νευρώνων:*

2.5. Επιλογή κέντρων για την εκπαίδευση του κρυφού (RBF) στρώματος

Όσον αφορά την επιλογή των κέντρων στην εκπαίδευση του κρυφού (RBF) στρώματος, υπάρχουν διάφορες μέθοδοι. Στη συνέχεια θα αναλύσουμε δύο από αυτούς, οι οποίοι είναι ο κανόνας των **Κ- μέσων** (k-means) και **τυχαία**.

a) Κανόνας των **Κ-μέσων**

Το **clustering** (ομαδοποίηση) αποτελεί τον επικρατέστερο τρόπο επιλογής κέντρων. Η συνάρτηση "*kmeans(X, n_clusters, random_state = None, max_iters = 100)*" υλοποιεί την μέθοδο των Κ-Μέσων και προσπαθεί να διαχωρίσει τον χώρο του συνόλου δεδομένων X σε ένα πλήθος από μη επικαλυπτόμενες περιοχές (ομάδες) ίσες με τον αριθμό των κέντρων (clusters) τα οποία ζητάω, έτσι ώστε κάθε δείγμα που βρίσκεται εντός της περιοχής που αφορά ένα συγκεκριμένο κέντρο, σημαίνει ότι βρίσκεται πιο μακριά από τα υπόλοιπα κέντρα:

- "X": Ο πίνακας με τα δεδομένα εισόδου, όπου κάθε γραμμή είναι ένα δείγμα
- "*n_clusters*": Ο αριθμός των κέντρων των clusters
- "*random_state*": Αφορά τη ρύθμιση του τυχαίου σπόρου (seed) για την αναπαραγωγιμότητα των αποτελεσμάτων
- "*max_iters*": Ο μέγιστος αριθμός επαναλήψεων εκτέλεσης του βρόχου

```python
# K-Means
def kmeans(X, n_clusters, random_state=None, max_iters=100):
    if random_state:
        np.random.seed(random_state)
    centers = X[np.random.choice(X.shape[0], n_clusters, replace=False)]
    for _ in range(max_iters):
        distances = np.linalg.norm(X[:, None] - centers[None, :], axis=2)
        labels = np.argmin(distances, axis=1)
        new_centers = np.array([X[labels == k].mean(axis=0) for k in range(n_clusters)])
        if np.all(centers == new_centers):
            break
        centers = new_centers
    return centers
```

Αναλυτικότερα, ο αλγόριθμος λειτουργεί ως εξής:

- Επιλέγονται τυχαία "*n_clusters*" σημεία από τα δεδομένα ($X.shape[0]$ το πλήθος τους) ως αρχικά κέντρα "*centers*" των clusters. Χρησιμοποιείται η "*np.random.choice*" για να εξασφαλίσει ότι δεν επιλέγουμε το ίδιο σημείο δύο φορές (χρησιμοποιώντας "*replace=False*").

- Γίνεται χρήση ενός βρόχου "*for*" με μέγιστο αριθμό επαναλήψεων "*max_iters*", ώστε ο αλγόριθμος να σταματήσει σε περίπτωση που δεν συγκλίνει. Μέσα σε αυτόν, υπολογίζεται η Ευκλείδεια απόσταση μεταξύ κάθε σημείου δεδομένων και των κέντρων με χρήση της "*np.linalg.norm*" και αποθηκεύεται στον πίνακα "*distances*" μεγέθους $N \times K$, όπου N ο αριθμός των δειγμάτων και K ο αριθμός των clusters. Το $[:, None]$ προσθέτει από μία διάσταση στους πίνακες με σκοπό να τους κάνει συμβατούς σε μέγεθος για αφαίρεση, με σχήμα $N \times K \times D$, όπου D το πλήθος των χαρακτηριστικών.
  Έπειτα, τα δείγματα κατηγοριοποιούνται στο cluster του πλησιέστερου κέντρου βάσει της απόστασης τους, με χρήση της "*np.argmin*"και αποθηκεύονται στην μεταβλητή "*labels*". Τα νέα κέντρα υπολογίζονται ως ο μέσος όρος των δειγμάτων που ανήκουν σε κάθε cluster και αποθηκεύονται στην μεταβλητή "*new_centers*" (χρησιμοποιώντας τη συνθήκη *"labels==k"* για να φιλτράρουμε τα σημεία που ανήκουν στο cluster k). Αν τα νέα κέντρα είναι ίδια με τα προηγούμενα, η διαδικασία σταματά (σημάδι σύγκλισης).
- Επιστρέφονται τα τελικά κέντρα.
  (*Σημείωση: 100 ομάδες σημαίνει ότι το κρυφό μου στρώμα θα έχει 100 νευρώνες*)

b) **Τυχαία** επιλογή κέντρων

Η συνάρτηση "random_centers*(X, num_centers, random_state = None)*" υλοποιεί την τυχαία επιλογή κέντρων και είναι μια προφανώς απλούστερη και ταχύτερη μέθοδος, όπου επιλέγουμε τυχαία σημεία/δείγματα από τα δεδομένα για να χρησιμοποιηθούν ως κέντρα:

- "X": Ο πίνακας με τα δεδομένα εισόδου, όπου κάθε γραμμή είναι ένα δείγμα
- "*num_centers*": Ο αριθμός των κέντρων
- "*random_state*": Αφορά τη ρύθμιση του τυχαίου σπόρου (seed) για την αναπαραγωγιμότητα των αποτελεσμάτων

```
# Τυχαία επιλογή κέντρων
def random_centers(X, num_centers, random_state=None):
    if random_state:
        np.random.seed(random_state)
    indices = np.random.choice(X.shape[0], num_centers, replace=False)
    centers = X[indices]
    return centers
```

Αναλυτικότερα, ο κώδικας έχει ως εξής:

- Επιλέγονται "*num_centers*" τυχαίοι δείκτες από τα δεδομένα ($X.shape[0]$ το πλήθος τους) χωρίς αντικατάσταση με χρήση της "*np.random.choice*" και τα αντίστοιχα σημεία δεδομένων θεωρούνται τα κέντρα.
- Επιστρέφονται τα επιλεγμένα κέντρα "*centers*".

Δεν υπάρχουν επαναλήψεις ή διαδικασία βελτιστοποίησης, ενώ μπορεί να εξασφαλιστεί τα κέντρα να είναι ισοπίθανα εντός των κλάσεων (π.χ. αν έχω 100 κέντρα τότε να πάρω τυχαία 10 από κάθε μία από τις 10 κλάσεις της CIFAR-10).

2.6. Ενεργοποίηση του επιπέδου RBF (έξοδοι κρυφών νευρώνων)

Η συνάρτηση "*rbf_activation(X, centers, betas)*" υλοποιεί τη λειτουργία ενεργοποίησης του **RBF** (Radial Basis Function) επιπέδου:

- "X": Ο πίνακας με τα δεδομένα εισόδου, όπου κάθε γραμμή είναι ένα δείγμα
- "*centers*": Τα κέντρα για τους RBF νευρώνες, που έχουν προκύψει από έναν αλγόριθμο επιλογής κέντρων (βλ. *2.5, σελ.7*)
- "*betas*": Παράμετροι που καθορίζουν το "εύρος" κάθε RBF νευρώνα, δηλαδή των κέντρων

```python
# Ενεργοποιήσεις του του επιπέδου RBF
def rbf_activation(X, centers, betas):
    G = np.zeros((X.shape[0], centers.shape[0]))
    for i, center in enumerate(centers):
        distances = np.linalg.norm(X - center, axis=1)
        G[:, i] = np.exp(-betas[i] * distances ** 2)
    return G
```

Αναλυτικότερα, ο κώδικας έχει ως εξής:

- Αρχικοποιείται ο πίνακας "G" που θα περιέχει τις ενεργοποιήσεις, με "$X.shape[0]$" να είναι ο αριθμός των δειγμάτων και "$centers.shape[0]$" ο αριθμός των κέντρων (κρυφών νευρώνων).
- Μέσω ενός βρόχου "*for*", που επαναλαμβάνεται για κάθε κέντρο i, υπολογίζεται η Ευκλείδεια απόσταση κάθε δείγματος από το κέντρο i, με χρήση της "*np.linalg.norm*" κατά μήκος της διάστασης των χαρακτηριστικών ($axis = 1$).
  Έπειτα, υπολογίζονται οι ενεργοποιήσεις των κρυφών νευρώνων με χρήση μίας **Gaussian** συνάρτησης ακτινικού τύπου: $\boldsymbol{\varphi(x, c) = e^{-\frac{\|x-c\|^2}{2\sigma^2}}}$, όπου $\boldsymbol{beta = \frac{1}{2\sigma^2}}$. Αυτή δημιουργεί μία καμπύλη σχήματος 'καμπάνας' γύρω από κάθε κέντρο με μέγιστη τιμή ίση με 1 όταν $x = c$ και μειώνεται εκθετικά καθώς το δείγμα x απομακρύνεται από το κέντρο c πλησιάζοντας το 0 (επιστρέφει τιμές μεταξύ 0 και 1). Έτσι, μοντελοποιούνται οι μη γραμμικές σχέσεις μεταξύ των δεδομένων.
  Όσον αφορά την τιμή του $\boldsymbol{\beta}$ (**beta**) έχουμε ότι:
  i. **Μεγάλη** τιμή $\boldsymbol{\beta}$: Η εκθετική συνάρτηση πέφτει πολύ γρήγορα με την απόσταση και η ενεργοποίηση περιορίζεται μόνο σε σημεία πολύ κοντά στο κέντρο.
  ii. **Μικρή** τιμή $\boldsymbol{\beta}$: Η εκθετική συνάρτηση πέφτει πιο αργά και η ενεργοποίηση είναι πιο διάχυτη, επηρεάζοντας και πιο μακρινά σημεία.

- Επιστρέφεται ο πίνακας "G", που περιέχει την ενεργοποίηση κάθε RBF νευρώνα για κάθε δείγμα. Οι ενεργοποιήσεις αυτές, λειτουργούν ως τα χαρακτηριστικά που προκύπτουν από την προβολή των δεδομένων στους RBF νευρώνες.

2.7. One-Hot Encoding

Η συνάρτηση "*one_hot_encode(labels, num_classes)*" μετατρέπει τα labels σε one-hot encoded διανύσματα, όπου για την περίπτωση μας είναι διανύσματα 10 στοιχείων (# of classes = 10) που είναι γεμάτα με μηδενικά εκτός από τον δείκτη της κλάσης στην οποία ανήκει το δείγμα (στόχος). Επομένως, είναι κατάλληλη για προβλήματα ταξινόμησης, ειδικά με την παράλληλη χρήση της SoftMax η οποία παράγει πιθανότητες για κάθε κατηγορία, καθώς επιτρέπει τη σωστή σύγκριση της εξόδου του μοντέλου με τις πραγματικές κατηγορίες (κλάσεις). Έτσι, επιστρέφει έναν πίνακαν με διαστάσεις $len(labels) \times num\_classes$, όπου κάθε γραμμή είναι το one-hot encoded διάνυσμα της αντίστοιχης κατηγορίας.

```
# Το στοιχείο του πίνακα που αντιστοιχεί στο label είναι 1, ΕΝΩ όλα τα άλλα 0
def one_hot_encode(labels, num_classes):
    return np.eye(num_classes)[labels] # Δημιουργία ταυτοτικού πίνακα
```

2.8. SoftMax

Η συνάρτηση ενεργοποίησης **SoftMax** που ορίζεται ως: $\boldsymbol{SoftMax_i(x)} = \frac{e^{x_i}}{\sum_j^K e^{x_j}}$, όπου x_i η είσοδος της SoftMax για την κατηγορία i, με K το πλήθος των κατηγοριών, χρησιμοποιείται στο επίπεδο εξόδου σε συνδυασμό με την Cross-Entropy Loss Function (βλ. *2.9, σελ.11*) και εξασφαλίζει ότι η έξοδος του μοντέλου μπορεί να ερμηνευθεί ως πιθανότητα για κάθε κατηγορία, ώστε το άθροισμα ένας να είναι ίσο με 1. Η κατηγορία με τη μεγαλύτερη πιθανότητα είναι η τελική πρόβλεψη (δηλαδή ένας νευρώνας είναι σωστός κάθε φορά).

```
# Η συνάρτηση ενεργοποίησης του επιπέδου εξόδου
def softmax(x):
    exp_x = np.exp(x - np.max(x, axis=1, keepdims=True)) # Αποφυγή Overflow
    return exp_x / np.sum(exp_x, axis=1, keepdims=True)
```

2.9. Cross-Entropy Loss function

Για τον υπολογισμό της συνάρτησης απώλειας χρησιμοποιείται η **Cross-Entropy Loss**, που είναι κατάλληλη για προβλήματα ταξινόμησης με SoftMax (βλ. *2.8, σελ.10*) στο επίπεδο εξόδου και ορίζεται ως: $\boldsymbol{L = \frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{ij} \cdot \log(y_{pred,ij})}$, όπου N("$Y_{true}.shape[0]$") το πλήθος των δειγμάτων, C ο αριθμός των κατηγοριών, y_{ij} η πραγματική πιθανότητα (ή **one-hot encoded**) για το i-οστό δείγμα και την j-οστή κατηγορία και $y_{pred,ij}$ η προβλεπόμενη πιθανότητα για την ίδια κατηγορία. Επιπλέο, προστίθεται το $1e-9$ για αποφυγή του λογαρίθμου του μηδενός (ο οποίος είναι άπειρος). Η SoftMax μετατρέπει τις εξόδους του δικτύου σε πιθανότητες που αθροίζονται σε 1. Η Cross-Entropy Loss αξιολογεί πόσο καλά αυτές οι πιθανότητες συμφωνούν με τις πραγματικές ετικέτες και αν η πρόβλεψη πλησιάζει την πραγματική κατηγορία (υψηλή πιθανότητα για το σωστό label), το loss είναι μικρό (επιβραβεύει τις σωστές προβλέψεις), ενώ αν η πρόβλεψη αποκλίνει από την πραγματική κατηγορία, το loss είναι μεγάλο (τιμωρεί τις λανθασμένες).

```
# Η συνάρτηση απώλειας
def cross_entropy_loss(Y_true, Y_pred):
    return -np.sum(Y_true * np.log(Y_pred + 1e-9)) / Y_true.shape[0]
```

2.10. Υπολογισμός ακρίβειας

Η συνάρτηση *"compute_accuracy"* υπολογίζει την ακρίβεια (accuracy) του μοντέλου, εκφρασμένη ως ποσοστό:

- ***"true_labels": Πίνακας με τις πραγματικές ετικέτες των δειγμάτων του test set.***
- *"predictions": Πίνακας με τις προβλέψεις του μοντέλου για τα αντίστοιχα δείγματα.*

```
# Συνάρτηση για τον υπολογισμό της ακρίβειας
def compute_accuracy(true_labels, predictions):
    return np.mean(true_labels == predictions) * 100
```

Αναλυτικότερα, το "*true_labels == predictions*" δημιουργεί έναν Boolean array, όπου κάθε στοιχείο είναι $True$ αν η πρόβλεψη είναι σωστή, διαφορετικά είναι False. Η "*np.mean*" υπολογίζει τον μέσο όρο του Boolean array, όπου τα $True$ λογίζονται ως 1 και τα False ως 0. Έτσι, ο μέσος όρος δίνει το ποσοστό των σωστών προβλέψεων (ακρίβεια σε δεκαδική μορφή). Το αποτέλεσμα πολλαπλασιάζεται με 100 για να εκφραστεί ως ποσοστό.

2.11. Παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης

Η συνάρτηση *"display_examples(y_true, y_pred, num_examples)"* εμφανίζει χαρακτηριστικά παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης. Τα παραδείγματα εμφανίζονται ως one-hot encoded διανύσματα με χρήση της συνάρτησης *"one_hot_encode(labels, num_classes)"* της παραγράφου *2.7*, για την πρόβλεψη και την ετικέτα. Η μεταβλητή *"correct_indices"* υπολογίζει τους δείκτες των παραδειγμάτων όπου η πρόβλεψη του μοντέλου είναι σωστή (δηλ. η ετικέτα *"y_true"* είναι ίση με την πρόβλεψη *"y_pred"*). Αντίστοιχα υπάρχει η *"in*correct_*indices"* για τις εσφαλμένες προβλέψεις.

```
# Παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης
def display_examples(y_true, y_pred, num_examples):
    print("\nΣωστές Κατηγοριοποιήσεις:")
    correct_indices = np.where(y_true == y_pred)[0]
    for i in correct_indices[:num_examples]: # έως num_examples
        print(f"Παράδειγμα {i}:")
        print(f"Πραγματική Ετικέτα (One-hot): {one_hot_encode([y_true[i]], 10)[0]}")
        print(f"Πρόβλεψη (One-hot): {one_hot_encode([y_pred[i]], 10)[0]}")
        print("-" * 30)

    print("\nΛανθασμένες Κατηγοριοποιήσεις:")
    incorrect_indices = np.where(y_true != y_pred)[0]
    for i in incorrect_indices[:num_examples]:
        print(f"Παράδειγμα {i}:")
        print(f"Πραγματική Ετικέτα (One-hot): {one_hot_encode([y_true[i]], 10)[0]}")
        print(f"Πρόβλεψη (One-hot): {one_hot_encode([y_pred[i]], 10)[0]}")
        print("-" * 30)
```

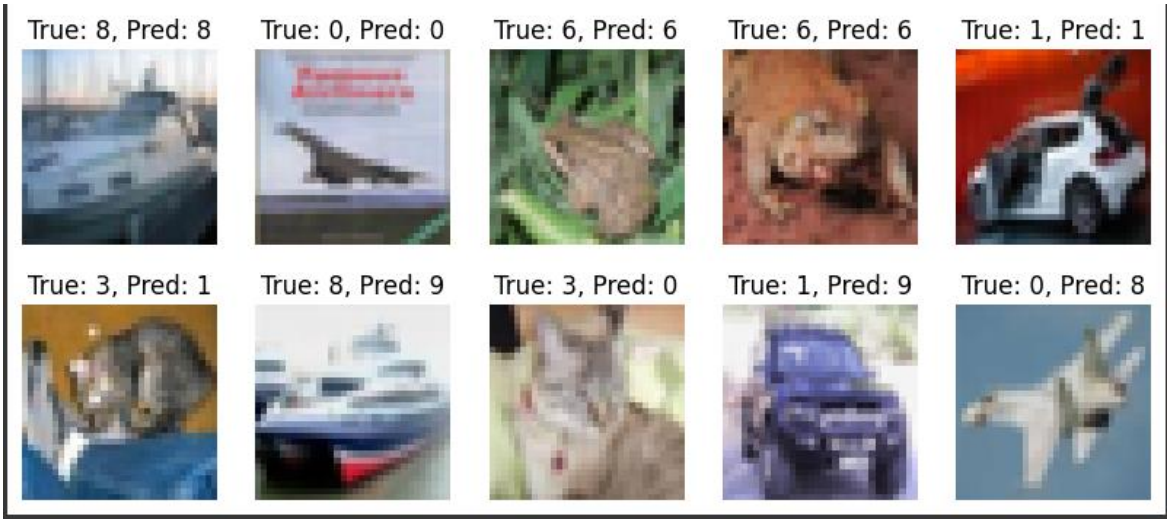
Ένα παράδειγμα εκτύπωσης ορθής και εσφαλμένης κατηγοριοποίησης των δειγμάτων του test set με δείκτες 0, 1, 2, 3, 4, 5 φαίνεται παρακάτω:

```
Παραδείγματα σωστής και λανθασμένης κατηγοριοποίησης:

Σωστές Κατηγοριοποιήσεις:
Παράδειγμα 1:
Πραγματική Ετικέτα (One-hot): [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
------------------------------
Παράδειγμα 2:
Πραγματική Ετικέτα (One-hot): [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
------------------------------
Παράδειγμα 5:
Πραγματική Ετικέτα (One-hot): [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
------------------------------

Λανθασμένες Κατηγοριοποιήσεις:
Παράδειγμα 0:
Πραγματική Ετικέτα (One-hot): [0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
------------------------------
Παράδειγμα 3:
Πραγματική Ετικέτα (One-hot): [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
------------------------------
Παράδειγμα 4:
Πραγματική Ετικέτα (One-hot): [0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
Πρόβλεψη (One-hot): [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]
------------------------------
```

Σε περίπτωση που δεν είχε πραγματοποιηθεί η κανονικοποίηση που παρουσιάζεται στην παράγραφο *2.3 (διαίρεση των δειγμάτων με 255.0)* και ανασχηματίζονταν τα δεδομένα στην αρχική τους κατάσταση $(32 \times 32 \times 3)$, θα μπορούσαν να παρουσιαστούν παραδείγματα κατηγοριοποιήσεων και με εικόνες με χρήση της βιβλιοθήκης "*matplotlib.pyplot*". Στο dataset CIFAR-10 οι κλάσεις είναι 10 συνολικά και έχουν τις εξής ετικέτες:

0: Airplane, **1**: Automobile, **2**: Bird, **3**: Cat, **4**: Deer, **5**: Dog. **6**: Frog, **7**: Horse, 8: Ship, **9**: Truck

Ένα παράδειγμα εμφάνισης των εικόνων φαίνεται παρακάτω:



2.12. Κύριο πρόγραμμα

Στην παρούσα αναφορά, παρουσιάζονται δύο διαφορετικές υλοποιήσεις ενός δικτύου Radial Basis Function (RBF) με στόχο την κατηγοριοποίηση δεδομένων από το σύνολο CIFAR-10. Οι **δύο προσεγγίσεις** διαφοροποιούνται τόσο στον **τρόπο επιλογής** των **κέντρων** όσο και στη **μέθοδο εκπαίδευσης** του μοντέλου.

Η **πρώτη** υλοποίηση χρησιμοποιεί τον αλγόριθμο **K-Means** για την επιλογή κέντρων και τη μέθοδο **backpropagation** με **gradient descent** (γενικευμένη Delta Rule) για την εκπαίδευση των βαρών του δικτύου. Η **δεύτερη** υλοποίηση βασίζεται σε **τυχαία επιλογή κέντρων** και χρησιμοποιεί τη μέθοδο των **ελαχίστων τετραγώνων** (**least squares**) με **κανονικοποίηση** (regularization) για την εκτίμηση των βαρών.

A. Πρώτη Υλοποίηση: K-Means και Backpropagation

Η συνάρτηση *"main()"* αποτελεί το κύριο πρόγραμμα για την εκπαίδευση και αξιολόγηση ενός **RBFNN** πάνω στο dataset **CIFAR-10**. Αναλυτικότερα:

I. Φορτώνονται τα δεδομένα εκπαίδευσης ("X_*train"*) και ελέγχου ("X_*test"*) μαζί με τις αντίστοιχες ετικέτες ("Y_*train "*, "Y_*test"*) από τη συνάρτηση *"load_cifar10()"* (βλ. *2.3, σελ.4*).

Εφαρμόζεται **PCA** στα δεδομένα εκπαίδευσης (βλ. *2.4, σελ.5*) διατηρώντας το 91% της συνολικής διακύμανσης και χρησιμοποιούνται τα επιλεγμένα ιδιοδιανύσματα ("*selected_eigenvectors*") και ο μέσος όρος ("*train_data_mean*") των δεδομένων εκπαίδευσης για την εφαρμογή PCA στα δεδομένα ελέγχου.

```
X_train, Y_train, X_test, Y_test = load_cifar10_data()

print("Εφαρμογή PCA στα δεδομένα εκπαίδευσης...")
X_train_pca, selected_eigenvectors, X_train_mean = pca(X_train, 0.91)

# Μετασχηματισμός του test set με την ίδια PCA
X_test_centered = X_test - X_train_mean
X_test_pca = np.dot(X_test_centered, selected_eigenvectors)
```

II. Γίνεται εκκίνηση του χρονομέτρου, ώστε να μετρηθεί ο συνολικός χρόνος εκπαίδευσης και ακολουθεί η **εκπαίδευση** του **κρυφού στρώματος** (**RBF**)**,** όπου εφαρμόζεται ο αλγόριθμος **K-Means** (βλ. *2.5, σελ.7*) για την επιλογή των κέντρων των RBF νευρώνων.

Για τον καθορισμό του εύρους "***sigma***" (σ) των Gaussian RBFs χρησιμοποιείται ο τύπος: $\sigma = \frac{d}{\sqrt{2 \cdot p}}$ όπου d είναι η μέγιστη απόσταση μεταξύ των κέντρων και p το πλήθος των κέντρων/νευρώνων. Με βάση αυτό υπολογίζονται και τα "*betas*".

Υπολογίζονται οι **ενεργοποιήσεις RBF** για κάθε είσοδο (train και test) και κάθε κέντρο. Αυτή η διαδικασία μετατρέπει τα δεδομένα σε μία μη-γραμμική αναπαράσταση.

```
start_time = time.time() # εκκίνηση χρονομέτρου
# Εκπαίδευση κρυφού στρώματος (RBF)
n_rbf_neurons = 100 # αριθμός των κρυφών νευρώνων/κέντρων
print("Εφαρμογή k - μέσων για την επιλογή των κέντρων...")
centers = kmeans(X_train_pca, n_rbf_neurons, random_state=0)

# Υπολογισμός του σ και των βήτα
max_distance = np.max([np.linalg.norm(c1 - c2) for i, c1 in enumerate(centers) for j, c2 in enumerate(centers) if i != j])
sigma = max_distance / np.sqrt(2 * n_rbf_neurons)
betas = np.full(n_rbf_neurons, 1 / (2 * sigma ** 2))

# Υπολογισμός ενεργοποιήσεων RBF
G_train = rbf_activation(X_train_pca, centers, betas)
G_test = rbf_activation(X_test_pca, centers, betas)
```

III. Οι τιμές των ενεργοποιήσεων RBF κανονικοποιούνται, διαιρώντας κάθε γραμμή του πίνακα "*G_train*" με την μέγιστη τιμή της γραμμής, ώστε κάθε γραμμή να έχει

μέγιστη τιμή το 1, με στόχο τον περιορισμό της δυναμικής περιοχής των τιμών και την σταθεροποίηση της εκπαίδευσης, αν και συνεχίζει να υπάρχει κάποια ανισορροπία. Για παράδειγμα:

```
G_train πριν την κανονικοποίηση: min=3.7284153313769036e-62, max=1.0, mean=0.0011610487667294636
G_test πριν την κανονικοποίηση: min=6.702104437367101e-62, max=0.6964124564527928, mean=0.0011628886561409928
G_train μετά την κανονικοποίηση: min=8.948795388665837e-60, max=1.0, mean=0.033246293020228454
G_test μετά την κανονικοποίηση: min=3.172922632839031e-59, max=1.0, mean=0.034320669826215774
```

Έτσι, τα βάρη και το learning rate συγκλίνουν πιο ομαλά κατά τη διάρκεια της εκπαίδευσης (τα losses είναι μικρότερα).

Προστίθεται ένας επιπλέον όρος "$bias$" στα δεδομένα εισόδου που βοηθά στην ευελιξία του μοντέλου και μαθαίνεται δυναμικά κατά την εκπαίδευση.

Το διάνυσμα ετικετών μετατρέπεται σε **one-hot** μορφή (π.χ., [0, 0, 1, 0, ...] για την κατηγορία 2) για κάθε κατηγορία/κλάση, ώστε να είναι συμβατό με την έξοδο του νευρωνικού δικτύου.

```python
# Normalization στο G
G_train /= np.max(G_train, axis=1, keepdims=True)
G_test /= np.max(G_test, axis=1, keepdims=True)

# Προσάρτηση bias στο G
G_train_aug = np.hstack([G_train, np.ones((G_train.shape[0], 1))])
G_test_aug = np.hstack([G_test, np.ones((G_test.shape[0], 1))])

# One hot encoding
Y_train_one_hot = one_hot_encode(Y_train, 10)
Y_test_one_hot = one_hot_encode(Y_test, 10)
```

IV. Παρακάτω έχουμε την αρχικοποίηση των παραμέτρων και των μεταβλητών για την **εκπαίδευση** του **εξωτερικού στρώματος** μέσω backpropagation. Τα **βάρη** αρχικοποιούνται με τη μέθοδο **Xavier** (Normal), ώστε να διατηρηθεί η σταθερότητα των διαβαθμίσεων κατά την εκπαίδευση.

```python
# Αρχικοποίηση υπεραπαραμέτρων και μεταβλητών
limit = np.sqrt(6 / (G_train_aug.shape[1] + 10)) # Xavier uniform
weights = np.random.uniform(-limit, limit, size=(G_train_aug.shape[1], 10))
learning_rate = 0.001
epochs = 100
best_loss = float('inf')
wait = 2
no_improvement_epochs = 0
lr_decay = 0.7
num_examples = 3  # Αριθμός παραδειγμάτων για εμφάνιση
```

V. Ο βρόχος *"for"* διατρέχει τις εποχές (epochs) και μέσα σε αυτόν υπολογίζονται οι **έξοδοι** του δικτύου μέσω του εσωτερικού γινομένου των ενεργοποιήσεων των RBF νευρώνων, τις οποίες έχουμε κανονικοποιήσει και στις οποίες έχουμε προσαρτήσει τους όρους $bias$, με τα βάρη. Οι έξοδοι μετατρέπονται σε **πιθανότητες** μέσω της συνάρτησης SoftMax (βλ. *2.8, σελ.10*), υπολογίζονται τα **σφάλματα** και η **απώλεια** μέσω της συνάρτησης Cross-Entropy Loss (βλ. *2.9, σελ.11*) σε train και test set και **ενημερώνονται** τα **βάρη** μέσω gradient descent.

```
# Εκπαίδευση του εξωτερικού στρώματος μέσω BP
for epoch in range(epochs):
    # Υπολογισμός εξόδων
    train_outputs = np.dot(G_train_aug, weights)
    test_outputs = np.dot(G_test_aug, weights)

    # Μετατροπή των εξόδων σε πιθανότητες (SoftMax)
    train_outputs_prob = softmax(train_outputs)
    test_outputs_prob = softmax(test_outputs)

    # Υπολογισμός σφαλμάτων
    train_error = Y_train_one_hot - train_outputs_prob
    test_error = Y_test_one_hot - test_outputs_prob

    # Υπολογισμός απώλειας (Cross-Entropy)
    train_loss = cross_entropy_loss(Y_train_one_hot, train_outputs_prob)
    test_loss = cross_entropy_loss(Y_test_one_hot, test_outputs_prob)

    # Ενημέρωση βαρών με regularization
    weights += np.dot(G_train_aug.T, train_error) * learning_rate
```

Ακόμη μέσα στον βρόχο "*for*" εφαρμόζεται δυναμική προσαρμογή του **ρυθμού μάθησης** "*learning_rate*", σύμφωνα με την οποία αν το "*train_loss*" της εποχής είναι μικρότερο από το "*best_loss*", αυτό ενημερώνεται, και ο μετρητής "*no_improvement_epochs*" μηδενίζεται. Αν το "*train_loss*" δεν βελτιωθεί για αριθμό εποχών ίσο με "*wait*" και $learning\_rate > 10^{-6}$, το "*learning_rate*" μειώνεται πολλαπλασιάζοντάς το με "*lr_decay*".
Υπολογίζονται οι **προβλέψεις** και τα **ποσοστά επιτυχίας** στα στάδια της εκπαίδευσης, και του ελέγχου. Τα "*train_outputs_prob*" και "*test_outputs_prob*" είναι σε μορφή one-hot encoding (βλ. *2.7, sel.10*) και η "*np.argmax*" εξάγει τον δείκτη της κατηγορίας για κάθε δείγμα αποθηκεύοντας τον στο "*train_predictions*" και "*test_predictions*" αντίστοιχα. Ύστερα, υπολογίζεται η **ακρίβεια** με χρήση της συνάρτησης "*compute_accuracy*" (βλ. *2.10, σελ.11*) και εκτυπώνονται τα αποτελέσματα για κάθε εποχή.

Εκτός του βρόχου "*for*" πλέον, τερματίζει το χρονόμετρο και προβάλλονται παραδείγματα ορθής και εσφαλμένης κατηγοριοποίησης με χρήση της συνάρτησης "*display_examples*" (βλ. *2.11, σελ.12*).

```
    # Αν το train loss βελτιώθηκε, ενημέρωση του καλύτερου loss
    if train_loss < best_loss:
        best_loss = train_loss
        no_improvement_epochs = 0
    else:
        no_improvement_epochs += 1
        # Μείωση του learning rate αν δεν βελτιωθεί για αριθμό εποχών = wait
        if no_improvement_epochs >= wait and learning_rate > 1e-6:
            learning_rate *= lr_decay
            no_improvement_epochs = 0  # Επαναφορά του μετρητή
            print(f"Epoch {epoch + 1}/{epochs}: Reducing learning rate to {learning_rate:.6f}")

    # Υπολογισμός προβλέψεων και ακρίβειας
    train_predictions = np.argmax(train_outputs_prob, axis=1)
    test_predictions = np.argmax(test_outputs_prob, axis=1)
    train_accuracy = compute_accuracy(Y_train, train_predictions)
    test_accuracy = compute_accuracy(Y_test, test_predictions)

    print(f"Epoch {epoch + 1}/{epochs}: Train Loss = {train_loss:.4f}, Test Loss = {test_loss:.4f}, "
          f"Train Accuracy = {train_accuracy:.4f}, Test Accuracy = {test_accuracy:.4f}")

total_training_time = time.time() - start_time
print(f"Συνολικός Χρόνος Εκπαίδευσης: {total_training_time:.2f} δευτερόλεπτα")

# Εμφάνιση παραδειγμάτων σωστής και λανθασμένης κατηγοριοποίησης
print("\nΠαραδείγματα σωστής και λανθασμένης κατηγοριοποίησης:")
display_examples(Y_test, test_predictions, num_examples)
```

B. Δεύτερη Υλοποίηση: Random Centers και Least Squares

Η **φόρτωση** των **δεδομένων**, η **εφαρμογή** της **PCA**, ο **υπολογισμός** των **ενεργοποιήσεων** RBF, η **κανονικοποίηση** σε αυτές και η **προσάρτηση** του **bias** μέσα στην **κύρια συνάρτηση** "*main()*" γίνονται **πανομοιότυπα** με την **πρώτη υλοποίηση** (βλ. 2.12.Α, σελ. 14). Ωστόσο, με την **δεύτερη** υλοποίηση, αν έχω όλα τα δεδομένα μου διαθέσιμα μπορώ να αποφύγω τον επαναληπτικό τύπο, με εκπαίδευση για ενημέρωση των βαρών σε κάθε εποχή, που εφαρμόστηκε στην πρώτη υλοποίηση και να χρησιμοποιήσω την **μέθοδο** των **ελαχίστων τετραγώνων** με **κανονικοποίηση** για άμεση ενημέρωση των βαρών.

Έτσι, ορίστηκε επιπρόσθετα η συνάρτηση "*least_squares_with_regularization(G, Y, lambda_reg)*", όπου G ο πίνακας που περιέχει τις ενεργοποιήσεις μετά το κρυφό επίπεδο (RBF) με κάθε γραμμή να αντιστοιχεί δείγμα και κάθε στήλη σε χαρακτηριστικό, Y ο πίνακας των labels και "$lambda\_reg$" ο όρος κανονικοποίησης.

```
def least_squares_with_regularization(G, Y, lambda_reg=1e-3):
    I = np.eye(G.shape[1])
    I[-1, -1] = 0  # Για να μην κανονικοποιηθεί το Bias
    W = np.dot(np.linalg.inv(np.dot(G.T, G) + lambda_reg * I), np.dot(G.T, Y))
    return W
```

Αναλυτικότερα:

- Δημιουργείται ένα μοναδιαίος πίνακας I διαστάσεων $G.shape[1] \times G.shape[1]$ (ίδιο πλήθος χαρακτηριστικών όπως στο G). Αυτός ο πίνακας χρησιμοποιείται για να εφαρμόσουμε την κανονικοποίηση, προσθέτοντας τον όρο λI στην μήτρα συσχέτισης των ενεργοποιήσεων $G^T G$.
- Το "$I[-1.-1] = 0$" εξαιρεί το τελευταίο στοιχείο του I από την κανονικοποίηση, καθώς αντιστοιχεί στον όρο bias (που έχει προστεθεί ως στήλη 1 στον πίνακα G). Με αυτή την αλλαγή, ο όρος regularization δεν εφαρμόζεται στο bias.
- Υπολογίζεται και επιστρέφεται το διάνυσμα βαρών W σύμφωνα με τη μέθοδο των **ελαχίστων τετραγώνων με κανονικοποίηση**: $(\boldsymbol{G^T G + \lambda I})^{-1} \boldsymbol{G^T Y}$

Η συνάρτηση αυτή καλείται μέσα στην "*main()*" ως :

```
# Least Squares με regularization
W = least_squares_with_regularization(G_train_aug, Y_train_one_hot)
```

Με "G_train_aug" να είναι ο πίνακας των ενεργοποιήσεων RBF μετά την κανονικοποίηση και την προσθήκη της στήλης $bias$ και "$Y\_train\_one\_hot$" ο πίνακας με τις ετικέτες σε μορφή one-hot encoding (βλ. *2.7, σελ.10*). Ο πίνακας W έχει διαστάσεις $(αριθμός\ χαρακτηριστικών + 1) \times αριθμός\ κατηγοριών$.

Επιπρόσθετα, σε αυτήν την υλοποίηση εφαρμόζεται **τυχαία επιλογή κέντρων** (βλ. *2.5, σελ.8*) ενώ διαφοροποιείται και ο τρόπος υπολογισμού της παραμέτρου "σ" (η οποία

χρησιμοποιείται στις Gaussian RBF), καθώς υπολογίζεται για κάθε κέντρο $c1$ ξεχωριστά, βασισμένη στις αποστάσεις από τα υπόλοιπα κέντρα.:

```
print("Εφαρμογή τυχαίας επιλογής των κέντρων...")
centers = random_centers(X_train_pca, n_rbf_neurons, random_state=0)

# Υπολογισμός σ για κάθε κέντρο
sigma_per_center = np.zeros(centers.shape[0])
for i, c1 in enumerate(centers):
    distances = [np.linalg.norm(c1 - c2) for j, c2 in enumerate(centers) if i != j]
    sigma_per_center[i] = np.mean(distances)

# Υπολογισμός betas για κάθε κέντρο
betas = 1 / (2 * sigma_per_center ** 2)
```

Πιο συγκεκριμένα:

- Αρχικοποιείται ο πίνακας "*sigma_per_center*" με μήκος όσο και το πλήθος των κέντρων.
- Με τον βρόχο "*for*" υπολογίζεται η ευκλείδεια απόσταση ("*np.linalg.norm*") για κάθε κέντρο $c1$ από όλα τα υπόλοιπα κέντρα $c2$, με $c1, c2$ να ανήκουν στον πίνακα "*centers*", που περιέχει όλα τα δείγματα που αποτελούν κέντρα. Το "*enumerate*" χρησιμοποιείται καθώς επιστρέφει την τιμή του κέντρου με τον δείκτη του, ο οποίος χρειάζεται για να αποφευχθεί το ίδιο το κέντρο στον υπολογισμό αποστάσεων. Οι αποστάσεις αποθηκεύονται σε μία λίστα καθώς διευκολύνει την εφαρμογή του ελέγχου "$if\ i\ != j$".

    Υπολογίζεται η μέση απόσταση του κέντρου $c1$ από όλα τα υπόλοιπα κέντρα που υπολογίστηκαν στο προηγούμενο βήμα και αποθηκεύεται στη θέση i του πίνακα "*sigma_per_center*".

Αυτή η μέση απόσταση θεωρείται ένας αντιπροσωπευτικός τρόπος για να καθοριστεί το "σ" για κάθε RBF νευρώνα/κέντρο, καθώς αντικατοπτρίζει τη γεωμετρική "γειτονιά" γύρω από το κάθε κέντρο.

3. Πειραματισμοί και αποτελέσματα

Στα πειράματα που ακολουθούν, μελετήθηκε η επίδραση διαφόρων παραμέτρων και μεθόδων στην απόδοση του Radial Basis Function Neural Network (RBFNN). Στόχος ήταν να εξεταστεί η ακρίβεια του μοντέλου, η ταχύτητα εκπαίδευσης και η γενίκευσή του σε δεδομένα ελέγχου. Επομένως, πραγματοποιήθηκαν πειράματα για διαφορετικούς **τρόπους επιλογής κέντρων** και **εκπαίδευσης**, καθώς και διάφορες τιμές των **παραμέτρων** εκπαίδευσης όπως "*sigma*", "*learning_rate*", "*epochs*", "*lambda_reg*", "*max_iters*" και αριθμό νευρώνων.

Σημείωση**: *Η πρώτη υλοποίηση της συνάρτησης "main()" με χρήση **Backpropagation** που παρουσιάζεται στην παρ. 2.12.Α, σελ.14 θα αναφέρεται ως **Α-Υλοποίηση** και η δεύτερη υλοποίηση με χρήση **Least Squares** που παρουσιάζεται στην παρ. 2.12.Β, σελ.17 θα αναφέρεται ως **Β-Υλοποίηση.

3.1. PCA

Για την **PCA** (βλ. *2.4, σελ.5*) επιλέχθηκε να διατηρηθεί 91% της διασποράς καθώς δεν παρατηρήθηκε αξιοσημείωτη διαφορά για μεγαλύτερα ποσοστά της διασποράς πέρα από τον αυξημένο χρόνο εκπαίδευσης. Συνεπώς η νέα διάσταση του χώρου των δεδομένων εισόδου έχει 114 συνιστώσες, ενώ αρχικά είχε 3072 ($32 \times 32 \times 3$) και μετά το κρυφό επίπεδο έχει διάσταση ίση με το πλήθος των κρυφών νευρώνων. Ένα παράδειγμα με διαφορετικά ποσοστά για την PCA και χρήση της **Β-Υλοποίησης** με **Τυχαία επιλογή κέντρων** όντας ταχύτερη σε εκτέλεση ($\#\ \nu\epsilon\upsilon\rho\acute{\omega}\nu\omega\nu\ =\ 1000$ και $lambda\_reg\ =\ 1e-2$) φαίνεται παρακάτω:

PCA	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
0.91	54.58 sec	52.62 %	49.47 %	2.0833	2.0933
0.95	92.83 sec	52.63 %	49.63 %	2.0841	2.0962
0.99	281.39 sec	52.16 %	49.60 %	2.0889	2.1007

Με διατήρηση 95% της διασποράς η νέα διάσταση του χώρου έχει 217 συνιστώσες και με διατήρηση 99% έχει 658 συνιστώσες, για αυτό και αυξάνεται ο χρόνος εκπαίδευσης. Επιπλέον, είναι σημαντικό να αναφερθεί ότι με την μέθοδο K-Μέσων για την επιλογή των κέντρων, η χρήση μεγαλύτερων ποσοστών από 91% δεν λειτουργούσε ούτε για αριθμό κρυφών νευρώνων ίσο με 100.

3.2. Αριθμός νευρώνων και επιλογή κέντρων

Ο αριθμός των νευρώνων στο κρυφό επίπεδο του RBFNN, παίζει κρίσιμο ρόλο στην απόδοσή του. Οι νευρώνες λειτουργούν ως κόμβοι που επεξεργάζονται την πληροφορία εισόδου, επιτρέποντας στο δίκτυο να "μάθει" πολύπλοκα πρότυπα στα δεδομένα. Κάθε νευρώνας στο RBF αντιστοιχεί σε ένα "κέντρο" και εφαρμόζει μια μη γραμμική συνάρτηση (Gaussian RBF) που καθορίζει την απόκριση του σε σχέση με την απόσταση από τα δεδομένα εισόδου.

Η επιλογή του αριθμού των κρυφών νευρώνων αποτελεί μία από τις βασικές υπερπαραμέτρους που επηρεάζουν την ισορροπία μεταξύ ικανότητας γενίκευσης και υπερεκπαίδευσης (overfitting). Ένας μικρός αριθμός νευρώνων ενδέχεται να μην επαρκεί για την αναπαράσταση των δεδομένων με ακρίβεια, ενώ ένας υπερβολικά μεγάλος αριθμός μπορεί να οδηγήσει σε αυξημένες υπολογιστικές απαιτήσεις, υπερβολική πολυπλοκότητα του μοντέλου και χρήση μεγάλης ποσότητας μνήμης RAM, με ενδεχόμενη την εμφάνιση σφαλμάτων λόγω χρήσης όλης της διαθέσιμης μνήμης, που δεν επιλύονται με batch training.

Οι πειραματικές διαδικασίες περιλαμβάνουν την εξέταση διαφορετικών αριθμών νευρώνων, αξιολογώντας την απόδοση του δικτύου σε σχέση με την ακρίβεια κατηγοριοποίησης, τον χρόνο εκπαίδευσης και την κατανάλωση μνήμης:

- **<u>Α-Υλοποίηση</u>**
  Με χαρακτηριστικά:
  - PCA με 91% (βλ. *2.4, σελ.5)*
  - **Κ-Μέσων** για την επιλογή κέντρων με "*max_iters = 100*"(βλ. *2.5, σελ.7*)
  - Υπολογισμός μοναδικού "σ" (βλ. *2.12.Α, σελ.14*)
  - Κανονικοποίηση Row-wise Max στις ενεργοποιήσεις του κρυφού στρώματος (βλ. *2.12.Α, σελ.14*)
  - "*epochs*" = 100, αρχικό "*learning_rate" = 0.001*", "*lr_decay*" = 0.7 και "*wait*" = 2
  - Xavier Uniform αρχικοποίηση για τα βάρη
  - Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

# νευρώνων	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
50	212.57 sec	33.47 %	33.38 %	1.8663	1.8631
100	413.85 sec	35.32 %	35.64 %	1.8276	1.8355
200	ΣΦΑΛΜΑ λόγω μνήμης RAM				

Συνοψίζοντας, συμπεραίνουμε ότι:

- Ο **χρόνος εκπαίδευσης** αυξάνεται σημαντικά με τον αριθμό των νευρώνων. Αυτό είναι αναμενόμενο, καθώς η υπολογιστική πολυπλοκότητα του RBFNN εξαρτάται από τον αριθμό των κρυφών νευρώνων (επίπεδο RBF), ο οποίος καθιστά τον υπολογισμό ενεργοποιήσεων και κέντρων (σύμφωνα με Κ-Μέσων) πολύ απαιτητικό.

- Η **ακρίβεια** βελτιώνεται ελαφρώς καθώς αυξάνεται ο αριθμός των νευρώνων. Η βελτίωση αυτή δείχνει ότι το δίκτυο μαθαίνει περισσότερα χαρακτηριστικά του συνόλου εκπαίδευσης με τους επιπλέον νευρώνες.
- Το **Train Loss** μειώνεται καθώς αυξάνεται ο αριθμός των νευρώνων, γεγονός που δείχνει ότι το δίκτυο προσαρμόζεται καλύτερα στα δεδομένα εκπαίδευσης. Αντίστοιχα, μειώνεται και το **Test Loss**, υποδεικνύοντας ότι η γενίκευση του δικτύου βελτιώνεται ελαφρώς.
- Παρατηρείται **πρόβλημα** στη μνήμη **RAM** για 200 νευρώνες (γενικά 125 και πάνω) κατά την εκτέλεση των **K-Μέσων**, διότι η υπολογιστική διαδικασία για τον υπολογισμό της απόστασης κάθε δείγματος από κάθε κέντρο απαιτεί μνήμη που αυξάνεται γραμμικά με τον αριθμό των νευρώνων και των δεδομένων (δοκιμάστηκε batch training, αλλά δεν παρατηρήθηκαν διαφορές). Αυτό συμβαίνει λόγω των περιορισμένων υπολογιστικών πόρων που διαθέτω.

Με χαρακτηριστικά:

- **Τυχαία επιλογή κέντρων** (βλ. *2.5, σελ.8*)
- Τα υπόλοιπα χαρακτηριστικά πανομοιότυπα με προηγουμένως

# νευρώνων	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
50	9.01 sec	29.57 %	30.06 %	1.9782	1.9684
100	13.81 sec	31.08 %	31.05 %	1.9274	1.9349
200	26.33 sec	32.87 %	32.54 %	1.8932	1.8994
500	57.70 sec	31.62 %	31.23 %	1.9639	1.9719

Συνοψίζοντας, συμπεραίνουμε ότι:

- Για διακύμανση σε αριθμούς νευρώνων 50-200 υπάρχει μικρή βελτίωση στην ακρίβεια και μείωση στις απώλειες με την αύξηση των νευρώνων, γεγονός που δείχνει ότι το μοντέλο μαθαίνει καλύτερα τις σχέσεις των δεδομένων. Ωστόσο, παρά την αύξηση του αριθμού των νευρώνων στους 500, η ακρίβεια μειώνεται ελαφρώς και η απώλειες αυξάνονται. Αυτό πιθανώς οφείλεται σε υπερεκπαίδευση ή ακατάλληλη επιλογή των κέντρων, που προκαλεί υπερπροσαρμογή σε μη αντιπροσωπευτικά δεδομένα.
- Η αύξηση των νευρώνων δεν αποδίδει αναλογικά σε ακρίβεια, ενώ μπορεί να οδηγήσει σε υποβάθμιση απόδοσης για μεγαλύτερους αριθμούς
- Ο χρόνος εκπαίδευσης αυξάνεται γραμμικά με τον αριθμό των νευρώνων.
- Η **τυχαία επιλογή κέντρων** αποδεικνύεται **ταχύτερη** αλλά **λιγότερο αποδοτική** από τη μέθοδο **K-Μέσων** όσον αφορά την ακρίβεια, καθώς δεν εξασφαλίζει ότι τα κέντρα αντιπροσωπεύουν επαρκώς τα δεδομένα. Η ακρίβεια είναι περίπου 3-4% χαμηλότερη, ενώ παρατηρούνται υψηλότερα επίπεδα απώλειας.

- **<u>Β-Υλοποίηση</u>**

Με χαρακτηριστικά:

- PCA με 91% (βλ. *2.4, σελ.5)*
- **Τυχαία επιλογή κέντρων** (βλ. *2.5, σελ.8*)
- "*lamda_reg*" = 1e-2 (βλ. *2.12.B, σελ.17*)
- Υπολογισμός "σ" για κάθε κέντρο (βλ. *2.12.B, σελ.18*)
- Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

# νευρώνων	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
50	2.36 sec	38.00 %	38.07 %	2.1946	2.1941
100	6.79 sec	40.68 %	40.91 %	2.1734	2.1730
200	10.01 sec	44.16 %	43.98 %	2.1479	2.1487
500	35.17 sec	48.17 %	47.24 %	2.1166	2.1210
1000	53.35 sec	51.49 %	49.20 %	2.0932	2.1004
1500	72.03 sec	52.87 %	50.32 %	2.0834	2.0932
3000	183.22 sec	55.72 %	52.08 %	2.0648	2.0795
5000	413.20 sec	57.68 %	52.51%	2.0518	2.0700
7000	707.26 sec	58.90 %	53.14 %	2.0434	2.0638
8000+	ΣΦΑΛΜΑ λόγω μνήμης RAM				

Συνοψίζοντας, συμπεραίνουμε ότι:

- Έχουμε αύξηση της απόδοσης με τον αριθμό των νευρώνων. Το **Training Accuracy** βελτιώνεται σταθερά καθώς αυξάνεται ο αριθμός νευρώνων, φτάνοντας το 58.90% για 7000 νευρώνες. Το **Testing Accuracy** ακολουθεί την ίδια τάση μέχρι τους 3000 νευρώνες, αλλά οι βελτιώσεις φθίνουν για μεγαλύτερους αριθμούς, υποδεικνύοντας ενδεχόμενη **υπερεκπαίδευση** και ότι ο αριθμός νευρώνων πλησιάζει το σημείο κορεσμού για την απόδοση.
- Τα **Train Loss** και **Test Loss** μειώνονται με τον ίδιο ρυθμό με τα αντίστοιχα accuracies, καθώς αυξάνεται ο αριθμός νευρώνων, γεγονός που δείχνει βελτίωση της εκπαίδευσης.
- Ο **χρόνος εκπαίδευσης** αυξάνεται έντονα για μεγάλο αριθμό νευρώνων λόγω της πολυπλοκότητας του αλγόριθμου Least Squares και του υπολογισμού του "σ" για κάθε κέντρο.
- Για αριθμό νευρώνων άνω των 8000 η απαίτηση σε υπολογιστικούς πόρους υπερβαίνει τις δυνατότητες της υπολογιστικής μονάδας, οδηγώντας σε αποτυχία λόγω έλλειψης μνήμης RAM. Επομένως, υπάρχει ξεκάθαρο **όριο στους πόρους μνήμης**, που επηρεάζει την πρακτική χρήση της υλοποίησης με μεγάλο αριθμό νευρώνων. Η βέλτιστη επιλογή αριθμού νευρώνων πρέπει να βασίζεται σε **συμβιβασμό μεταξύ απόδοσης, χρόνου εκπαίδευσης και διαθέσιμων πόρων μνήμης**.

3.3. Υπολογισμός της παραμέτρου "σ"

Για τον υπολογισμό της παραμέτρου "σ" εφαρμόστηκαν δύο τρόποι:

- Υπολογισμός μοναδικού "σ" (βλ. *2.12.A, σελ.14*)
- Υπολογισμός "σ" για κάθε κέντρο (βλ. *2.12.B, σελ.18*)

Αυτοί δοκιμάστηκαν και για τις δύο υλοποιήσεις:

- **Α-Υλοποίηση**
  Με χαρακτηριστικά δικτύου:
  - PCA με 91% (βλ. *2.4, σελ.5)*
  - Κανονικοποίηση Row-wise Max στις ενεργοποιήσεις του κρυφού στρώματος (βλ. *2.12.A, σελ.14*)
  - "*epochs*" = 100, αρχικό "*learning_rate" = 0.001*", "*lr_decay*" = 0.7 και "*wait*" = 2
  - Xavier Uniform αρχικοποίηση για τα βάρη
  - Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

  I. Ένα παράδειγμα για τον υπολογισμό **μοναδικού** "**σ**":

Επιλογή κέντρων	**Χρόνος εκπαίδευσης**	**Training Accuracy**	**Testing Accuracy**	**Train Loss**	**Test Loss**
K-Μέσων	413.85 sec	35.32 %	35.64 %	1.8276	1.8355
Τυχαία	13.81 sec	31.08 %	31.05 %	1.9274	1.9349

```
Εφαρμογή k - μέσων για την επιλογή των κέντρων...
Τιμή sigma:
2.763959307155814
```

```
Εφαρμογή τυχαίας επιλογής κέντρων...
Τιμή sigma:
2.944257218276999
```

Συνοψίζοντας, συμπεραίνουμε ότι:

- Με χρήση **K-Μέσων** (υπολογιστικά απαιτητική διαδικασία) ο **χρόνος εκπαίδευσης** είναι σημαντικά μεγαλύτερος, οι **ακρίβειες** σε training και testing sets είναι αρκετά υψηλότερες και οι **απώλειες** είναι μικρότερες, (υποδεικνύοντας ότι το μοντέλο προσαρμόζεται καλύτερα στα δεδομένα), σε σύγκριση με τη χρήση **Τυχαίας επιλογής** κέντρων

II. Ένα παράδειγμα για τον υπολογισμό του "**σ**" **για κάθε κέντρο**:

Επιλογή κέντρων	**Χρόνος εκπαίδευσης**	**Training Accuracy**	**Testing Accuracy**	**Train Loss**	**Test Loss**
K-Μέσων	407.75 sec	22.73 %	23.14 %	10.7617	10.6595
Τυχαία	15.95 sec	22.09 %	21.97 %	6.3378	6.3487

```
Εφαρμογή k - μέσων για την επιλογή των κέντρων...
Τιμές sigma για κάθε κέντρο:
[11.16102982 15.35843501 10.16479961 13.20831151  9.90814871 11.46507219
 12.25742988 11.59868398 15.38760886 13.21857419 12.16429405 12.33363227
 19.44719618 15.02187122 10.44853002 19.88873598 14.72728448 13.28474577
 12.7239936  12.63620666 14.07594882 14.53699856 13.23462754 11.78172511
 18.10250378 16.63602801 11.1500311  10.00847062 11.68171274 21.93030948
 10.86842097 13.12752494 12.15730319 13.42491213 13.7031373  15.21504228
 11.78460602 11.54318613 13.46975656 10.71530602 11.95232296 16.24944173
 12.23785789 11.09659163 12.49536538 10.82410864 11.81015534 13.45998301
 13.94751956 16.9603919  13.03426911 17.61423369 11.85650885 13.63068797
 14.10090847 12.73473893 13.99544431 12.36081102 11.68651333 12.46565064
 12.94781142 11.01536243 15.59407606 13.13046167 16.1893077  14.82303596
 15.95599358 13.3688638  12.05816824 12.21992644 15.15725315 15.86833943
 17.33776463 12.25991938 12.2195072  12.83683123 10.87605602 11.36306942
 11.29533147 13.31135445 12.49443047 12.35661441 14.07193783 13.27053265
 13.2184816  15.47690835 12.14833518 10.29203656 13.24538943 12.48864584
 10.88134304 14.83519013 12.09190642 13.06195275 14.22953048 16.82662417
 12.36765556 12.87579976 13.05461941 14.13464093]
```

```
Εφαρμογή τυχαίας επιλογής κέντρων...
Τιμές sigma για κάθε κέντρο:
[17.04168091 15.53474396 21.2995506  21.71714642 18.75460663 21.64190562
 17.93443996 19.94893963 19.2250998  18.53757165 20.08653499 15.42821891
 17.10178238 15.90456274 22.39168211 15.99919373 17.18010903 16.119551
 22.78869263 20.65342683 22.66088054 16.69072155 16.23464222 16.55068059
 18.23706634 17.27154535 20.12091639 18.26344471 15.72898003 19.27181936
 15.30572718 18.0356507  17.9833447  21.08524541 14.30845423 15.49334967
 15.40049466 14.72921141 20.22607656 20.29816478 18.48557971 17.33491762
 19.43351164 16.68617008 18.77451972 19.66847241 15.50696728 16.286333
 16.07570111 18.19830901 17.71035716 21.67689475 17.83550522 19.88250219
 18.0330376  17.96435827 16.54532211 15.75803304 22.49329189 20.16514
 15.02578626 17.98416254 21.30272089 17.60345252 18.57902914 20.3949521
 16.86393427 18.55673127 16.66082435 17.65739185 24.97845534 24.08701615
 20.78762747 14.99928936 20.51729494 20.13534051 17.98525232 15.94127453
 17.27764624 19.83568334 17.50728625 19.91547097 16.89901437 18.97426972
 20.42904837 16.61337073 17.79685761 19.40329003 15.89532969 18.12145595
 18.05980954 16.11246061 18.7511307  16.34127159 23.07424824 15.36960911
 22.15312425 14.73101346 17.90012493 18.16583333]
```

Με το "*learning_rate*" και στις δύο περιπτώσεις να φτάνει στο όριο του στην epoch = 41, που είναι 0.000001.

Συνοψίζοντας, συμπεραίνουμε ότι:

- Με χρήση **Κ-Μέσων** (υπολογιστικά απαιτητική διαδικασία) ο **χρόνος εκπαίδευσης** είναι παρόμοιος με τη χρήση μοναδικού "σ", ωστόσο οι **ακρίβειες** σε training και testing sets είναι σημαντικά χαμηλότερες, γεγονός που δείχνει ότι η προσθήκη διαφορετικών τιμών "σ" ανά κέντρο ενδέχεται να αυξάνει την πολυπλοκότητα χωρίς να παρέχει βελτίωση. Οι **απώλειες** είναι εξαιρετικά υψηλές (~10), υποδεικνύοντας ότι το μοντέλο δεν καταφέρνει να μάθει αποτελεσματικά από τα δεδομένα.
- Με χρήση **Τυχαίας επιλογής κεντρων** ο **χρόνος εκπαίδευσης** παραμένει μικρός, όπως και στη χρήση μοναδικού "σ". Οι **ακρίβειες** είναι παρόμοιες με την χρήση Κ-Μέσων, ενώ οι **απώλειες** είναι αρκετά χαμηλότερες, αν και παραμένουν σε πολύ υψηλά επίπεδα.

Γενικά:

Για την **Α-Υλοποίηση**, η μέθοδος υπολογισμού **μοναδικού "σ"** παρουσιάζει καλύτερες επιδόσεις τόσο σε ακρίβεια όσο και σε απώλειες σε σύγκριση με τον υπολογισμό διαφορετικού "σ" ανά κέντρο. Αυτό συμβαίνει επειδή χρησιμοποιεί σταδιακή προσαρμογή βαρών μέσω της ελαχιστοποίησης του σφάλματος. Συνεπώς, ο χώρος χαρακτηριστικών που προκύπτει από στενά Gaussians βοηθά στη μάθηση σύνθετων,

μη γραμμικών σχέσεων. Επιπλέον, η επιλογή **K-Μέσων** για τα κέντρα, αν και πιο αργή, παρέχει καλύτερα αποτελέσματα από τη χρήση τυχαίων κέντρων.

- **<u>Β-Υλοποίηση</u>**

Με χαρακτηριστικά δικτύου:

- PCA με 91% (βλ. *2.4, σελ.5)*
- Αριθμός κρυφών νευρώνων ίσος με 100
- "*lamda_reg*" = 1e-3 (βλ. *2.12.Β, σελ.17*)
- Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

I. Ένα παράδειγμα για τον υπολογισμό μοναδικού "σ":

Επιλογή κέντρων	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
K-Μέσων	397.09 sec	34.84 %	34.77 %	2.1933	2.1904
Τυχαία	5.16 sec	29.44 %	30.16 %	2.2260	2.2248

```
Εφαρμογή k – μέσων για την επιλογή των κέντρων...
Τιμή sigma:
2.763959307155814
```

```
Εφαρμογή τυχαίας επιλογής των κέντρων...
Τιμή sigma:
2.4871521928106777
```

Συνοψίζοντας, συμπεραίνουμε ότι:

- Με χρήση **K-Μέσων** (υπολογιστικά απαιτητική διαδικασία) ο **χρόνος εκπαίδευσης** είναι σημαντικά μεγαλύτερος, οι **ακρίβειες** σε training και testing sets είναι υψηλότερες και οι **απώλειες** είναι μικρότερες, (υποδεικνύοντας ότι το μοντέλο προσαρμόζεται καλύτερα στα δεδομένα), σε σύγκριση με τη χρήση **Τυχαίας επιλογής** κέντρων

II. Ένα παράδειγμα για τον υπολογισμό του "σ" για κάθε κέντρο:

Επιλογή κέντρων	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
K-Μέσων	411.16 sec	41.26 %	41.58 %	2.1580	2.1581
Τυχαία	9.08 sec	40.85 %	41.56 %	2.1740	2.1732

```
Εφαρμογή k - μέσων για την επιλογή των κέντρων...
Τιμές sigma για κάθε κέντρο:
[10.55432585 14.23407256 13.57769075 12.64262124 15.6403448  14.67410682
 12.71218101 18.77387838 13.12045895 11.95752382 12.5389825  13.54961904
 13.62731165 13.026693   11.25912883 12.14469078 12.06388433 11.3937063
  9.78691694 16.05948582 11.03859237 12.99299546 12.20737608 11.78476581
 16.58990354 12.39760313 14.4846209  11.62308557 13.66356349 14.22242771
 13.57113331 13.26460196 17.14452747 12.9166145  13.19518736 13.09482822
 14.83904261 12.78824565 12.13932383 13.57887122 13.73363082 20.03879101
 13.04098871 12.41425449 12.76724769 15.51989546 15.06993517 11.55730194
 11.41060857 10.53067514 10.89591615 11.86294262 12.49375373 13.19828131
 13.06443146 15.16939211 13.87511435 13.36827631 15.601606   12.98816243
 15.59856368 10.47383289 12.76079091 17.66279119 11.64948195 15.20252491
 10.92411921 16.29101991 17.47613196 16.57817446 11.41186593 21.86975829
 12.8506582  15.95651951 14.69570783 10.97513065 13.1333785  15.70565894
 12.9469445  12.05585835 15.93976429 11.52444577 13.98188543 11.68256674
 13.53068024 10.44231977 10.86407147 12.09802779 14.07903345 12.4784222
 11.31322324 13.03635406 14.14244853 16.56701651 12.17514496 12.31861164
 11.06535034 12.25916661 12.67725341 19.29694491]
```

```
Εφαρμογή τυχαίας επιλογής των κέντρων...
Τιμές sigma για κάθε κέντρο:
[27.85698562 18.14630773 17.78539038 24.39585255 19.32814242 18.21721128
 16.44395272 17.78971535 18.75969379 24.93562805 18.07867788 19.5426988
 18.2771819  17.50393855 16.6973042  18.29992755 18.99557972 16.87842648
 18.03621532 15.53066138 21.25720488 17.11255507 22.47837682 18.6220935
 20.49366979 18.49141726 20.36407643 21.52887701 17.26097432 18.52527286
 18.62903    24.22015213 15.46563125 20.3366083  20.74789599 15.18068011
 17.04101453 17.76792836 16.71242937 18.39687446 19.9545779  21.66660075
 18.46765264 19.22574219 15.86473607 15.05309369 18.30698791 17.38288236
 19.95933573 22.2116413  18.29285705 16.04525021 21.20202684 16.99354604
 18.09915082 16.27108514 16.00829838 15.88794198 15.99172363 16.93301882
 17.21026133 17.74761982 17.01359622 21.66957729 25.04328133 19.20489093
 16.79787702 18.56632131 17.71050207 21.80719598 18.45312256 20.83109789
 17.84308347 25.47647633 18.98295507 17.75912908 20.2830455  18.41049585
 17.71667794 21.20472291 19.02605455 21.9703726  17.79151656 15.15625541
 17.28193196 16.83804446 19.63133008 17.60244185 20.64496039 17.26831554
 15.64442211 20.17917695 19.60785399 18.21358894 15.70292621 19.8076053
 18.68515694 19.95882911 18.03110442 22.03062002]
```

Συνοψίζοντας, συμπεραίνουμε ότι:

- Με χρήση **Κ-Μέσων** (υπολογιστικά απαιτητική διαδικασία) ο **χρόνος εκπαίδευσης** είναι παρόμοιος με τη χρήση μοναδικού "σ", ωστόσο οι **ακρίβειες** σε training και testing sets είναι σαφώς καλύτερες, γεγονός που δείχνει ότι ο υπολογισμός "σ" για κάθε κέντρο ενισχύει την απόδοση όταν τα κέντρα έχουν επιλεγεί προσεκτικά. Οι **απώλειες** είναι χαμηλότερες από την αντίστοιχη μέθοδο μοναδικού "σ", δείχνοντας βελτιωμένη προσαρμογή.
- Με χρήση **Τυχαίας επιλογής κεντρων** ο **χρόνος εκπαίδευσης** παραμένει αρκετά χαμηλός, όπως και στη χρήση μοναδικού "σ". Οι **ακρίβειες** και οι **απώλειες** είναι παρόμοιες με την χρήση Κ-Μέσων, γεγονός που υποδεικνύει ότι ο υπολογισμός διαφορετικού "σ" για κάθε κέντρο αντισταθμίζει τη χαμηλότερη ποιότητα των τυχαίων κέντρων.

Γενικά:

Για την **Β-Υλοποίηση**, η μέθοδος υπολογισμού **"σ" για κάθε κέντρο** ενισχύει σημαντικά την απόδοση του μοντέλου, ιδιαίτερα σε συνδυασμό με την **Τυχαία επιλογή κέντρων** που μπορεί να συνδυαστεί με μεγαλύτερο αριθμό κρυφών νευρώνων, σε σύγκριση με τον υπολογισμό μοναδικού "σ". Αυτό συμβαίνει επειδή υπολογίζει άμεσα τα βάρη ως λύση ενός προβλήματος ελαχιστοποίησης. Η προσέγγιση αυτή επωφελείται από έναν πιο πυκνό χώρο χαρακτηριστικών, όπου οι ενεργοποιήσεις επικαλύπτονται αρκετά ώστε να παρέχουν πληροφορίες για όλα τα δεδομένα.

3.4. Α-Υλοποίηση με χρήση Κ-Μέσων (hidden neurons = 100)

- **Ρυθμός μάθησης**

  Έχοντας δοκιμάσει αρκετές παραλλαγές όσον αφορά τον ρυθμό μάθησης για την *1η Υποχρεωτική Εργασία* του μαθήματος, στην οποία ο τρόπος εκπαίδευσης ήταν παρόμοιος, χρησιμοποιήθηκε **δυναμική προσαρμογή** του **learning rate** όπως παρουσιάζεται στην παράγραφο *2.12.Α, σελ.16.* Αυτό έγινε με σκοπό την βελτίωση της απόδοσης της εκπαίδευσης και την καλύτερη γενίκευση του μοντέλου, προσπαθώντας για αποφυγή αστάθειας ή μεγάλης αύξησης του train loss και αργής σύγκλισης λόγω χαμηλού ρυθμού μάθησης. Εν τέλει το δίκτυο εκπαιδεύτηκε για "*epochs*" = 100, με αρχικό "*learning_rate" = 0.001*", "*lr_decay*" = 0.7 και "*wait*" = 2 (χαρακτηριστικά της δυναμικής προσαρμογής του ρυθμού μάθησης, αν και τα αποτελέσματα ήταν παρόμοια για αρκετούς συνδυασμούς που δοκιμάστηκαν). Παρακάτω φαίνονται τα ποσοστά επιτυχίας και οι απώλειες στα στάδια της εκπαίδευσης και του ελέγχου για διάφορα παραδείγματα:

  i. Σταθερό learning rate = 0.001:

```
Epoch 1/100: Train Loss = 2.3223, Test Loss = 2.3226, Train Accuracy = 12.9020, Test Accuracy = 13.1500
Epoch 10/100: Train Loss = 8.7003, Test Loss = 8.7239, Train Accuracy = 21.2740, Test Accuracy = 21.5200
Epoch 20/100: Train Loss = 8.1577, Test Loss = 8.1778, Train Accuracy = 18.6480, Test Accuracy = 18.9200
Epoch 30/100: Train Loss = 8.1568, Test Loss = 8.1945, Train Accuracy = 23.0660, Test Accuracy = 23.2600
Epoch 40/100: Train Loss = 6.6475, Test Loss = 6.7126, Train Accuracy = 25.4060, Test Accuracy = 25.5900
Epoch 50/100: Train Loss = 6.0439, Test Loss = 6.0692, Train Accuracy = 19.2480, Test Accuracy = 20.0400
Epoch 60/100: Train Loss = 7.6475, Test Loss = 7.6933, Train Accuracy = 22.1100, Test Accuracy = 22.2700
Epoch 70/100: Train Loss = 6.0496, Test Loss = 6.0660, Train Accuracy = 25.0020, Test Accuracy = 25.2900
Epoch 80/100: Train Loss = 6.7688, Test Loss = 6.7994, Train Accuracy = 20.4020, Test Accuracy = 20.6000
Epoch 90/100: Train Loss = 5.9817, Test Loss = 6.0470, Train Accuracy = 27.9800, Test Accuracy = 27.5200
Epoch 100/100: Train Loss = 6.5147, Test Loss = 6.5795, Train Accuracy = 23.9940, Test Accuracy = 23.9800
```

  ii. Σταθερό learning rate = 0.01:

```
Epoch 1/100: Train Loss = 2.3273, Test Loss = 2.3298, Train Accuracy = 8.7820, Test Accuracy = 8.6100
Epoch 10/100: Train Loss = 17.9075, Test Loss = 17.8325, Train Accuracy = 11.1380, Test Accuracy = 11.3100
Epoch 20/100: Train Loss = 15.8690, Test Loss = 15.8351, Train Accuracy = 18.8000, Test Accuracy = 18.7200
Epoch 30/100: Train Loss = 15.8769, Test Loss = 15.8039, Train Accuracy = 18.3100, Test Accuracy = 19.0100
Epoch 40/100: Train Loss = 14.9978, Test Loss = 14.9232, Train Accuracy = 21.8980, Test Accuracy = 22.4000
Epoch 50/100: Train Loss = 15.4568, Test Loss = 15.4782, Train Accuracy = 21.0220, Test Accuracy = 21.0700
Epoch 60/100: Train Loss = 14.4374, Test Loss = 14.4039, Train Accuracy = 21.5360, Test Accuracy = 21.8400
Epoch 70/100: Train Loss = 16.2212, Test Loss = 16.2047, Train Accuracy = 17.9780, Test Accuracy = 17.9800
Epoch 80/100: Train Loss = 14.7584, Test Loss = 14.6880, Train Accuracy = 24.0020, Test Accuracy = 24.2700
Epoch 90/100: Train Loss = 14.8393, Test Loss = 14.8611, Train Accuracy = 22.1440, Test Accuracy = 21.8200
Epoch 100/100: Train Loss = 13.3434, Test Loss = 13.2990, Train Accuracy = 24.7760, Test Accuracy = 25.1200
```

  iii. Σταθερό learning rate = 0.0001:

```
Epoch 1/100: Train Loss = 2.3359, Test Loss = 2.3380, Train Accuracy = 8.9180, Test Accuracy = 8.5500
Epoch 10/100: Train Loss = 2.1570, Test Loss = 2.1524, Train Accuracy = 25.8640, Test Accuracy = 26.3100
Epoch 20/100: Train Loss = 2.0658, Test Loss = 2.0582, Train Accuracy = 30.2460, Test Accuracy = 31.0300
Epoch 30/100: Train Loss = 2.0131, Test Loss = 2.0043, Train Accuracy = 31.4620, Test Accuracy = 31.9300
Epoch 40/100: Train Loss = 1.9788, Test Loss = 1.9696, Train Accuracy = 32.2180, Test Accuracy = 32.6900
Epoch 50/100: Train Loss = 1.9545, Test Loss = 1.9452, Train Accuracy = 32.8020, Test Accuracy = 33.1900
Epoch 60/100: Train Loss = 1.9363, Test Loss = 1.9272, Train Accuracy = 33.0620, Test Accuracy = 33.4100
Epoch 70/100: Train Loss = 1.9222, Test Loss = 1.9134, Train Accuracy = 33.2580, Test Accuracy = 33.5400
Epoch 80/100: Train Loss = 1.9110, Test Loss = 1.9025, Train Accuracy = 33.4880, Test Accuracy = 33.6400
Epoch 90/100: Train Loss = 1.9017, Test Loss = 1.8937, Train Accuracy = 33.6120, Test Accuracy = 33.8300
Epoch 100/100: Train Loss = 1.8940, Test Loss = 1.8864, Train Accuracy = 33.7180, Test Accuracy = 33.9000
```

  iv. Δυναμική προσαρμογή του learning rate ("*epochs*" = 100, "*learning_rate" = 0.0001*", "*lr_decay*" = 0.5 και "*wait*" = 1)

```
Epoch 1/100: Train Loss = 2.3319, Test Loss = 2.3307, Train Accuracy = 8.5060, Test Accuracy = 8.5500
Epoch 3/100: Reducing learning rate to 0.000500
Epoch 4/100: Reducing learning rate to 0.000250
Epoch 5/100: Reducing learning rate to 0.000125
Epoch 6/100: Reducing learning rate to 0.000063
Epoch 7/100: Reducing learning rate to 0.000031
Epoch 10/100: Train Loss = 2.1960, Test Loss = 2.1913, Train Accuracy = 29.5140, Test Accuracy = 30.2400
Epoch 20/100: Train Loss = 2.0075, Test Loss = 2.0019, Train Accuracy = 31.3600, Test Accuracy = 32.0500
Epoch 30/100: Train Loss = 1.9527, Test Loss = 1.9470, Train Accuracy = 32.5420, Test Accuracy = 33.0400
Epoch 40/100: Train Loss = 1.9424, Test Loss = 1.9369, Train Accuracy = 32.7840, Test Accuracy = 33.6000
Epoch 50/100: Train Loss = 1.9365, Test Loss = 1.9313, Train Accuracy = 33.0080, Test Accuracy = 33.7600
Epoch 60/100: Train Loss = 1.9313, Test Loss = 1.9264, Train Accuracy = 33.1540, Test Accuracy = 33.8200
Epoch 70/100: Train Loss = 1.9265, Test Loss = 1.9219, Train Accuracy = 33.2320, Test Accuracy = 33.9400
Epoch 80/100: Train Loss = 1.9221, Test Loss = 1.9177, Train Accuracy = 33.3480, Test Accuracy = 33.9200
Epoch 90/100: Train Loss = 1.9181, Test Loss = 1.9139, Train Accuracy = 33.4400, Test Accuracy = 33.9800
Epoch 100/100: Train Loss = 1.9143, Test Loss = 1.9103, Train Accuracy = 33.4680, Test Accuracy = 33.9600
```

v. Δυναμική προσαρμογή του learning rate (“*epochs*“ = 100, “*learning_rate” = 0.001*”, “*lr_decay*” = 0.7 και “*wait*” = 2)

```
Epoch 1/100: Train Loss = 2.3238, Test Loss = 2.3202, Train Accuracy = 9.9320, Test Accuracy = 10.5900
Epoch 4/100: Reducing learning rate to 0.000700
Epoch 6/100: Reducing learning rate to 0.000490
Epoch 8/100: Reducing learning rate to 0.000343
Epoch 10/100: Reducing learning rate to 0.000240
Epoch 10/100: Train Loss = 2.6403, Test Loss = 2.6376, Train Accuracy = 23.9200, Test Accuracy = 24.7300
Epoch 20/100: Train Loss = 1.8926, Test Loss = 1.8867, Train Accuracy = 33.9240, Test Accuracy = 34.1900
Epoch 30/100: Train Loss = 1.8766, Test Loss = 1.8726, Train Accuracy = 34.2480, Test Accuracy = 34.2400
Epoch 40/100: Train Loss = 1.8667, Test Loss = 1.8641, Train Accuracy = 34.4060, Test Accuracy = 34.3500
Epoch 50/100: Train Loss = 1.8595, Test Loss = 1.8581, Train Accuracy = 34.5160, Test Accuracy = 34.3800
Epoch 60/100: Train Loss = 1.8541, Test Loss = 1.8537, Train Accuracy = 34.6020, Test Accuracy = 34.4700
Epoch 70/100: Train Loss = 1.8498, Test Loss = 1.8502, Train Accuracy = 34.6620, Test Accuracy = 34.5000
Epoch 80/100: Train Loss = 1.8463, Test Loss = 1.8475, Train Accuracy = 34.6900, Test Accuracy = 34.5900
Epoch 90/100: Train Loss = 1.8435, Test Loss = 1.8453, Train Accuracy = 34.7340, Test Accuracy = 34.5800
Epoch 100/100: Train Loss = 1.8411, Test Loss = 1.8435, Train Accuracy = 34.7720, Test Accuracy = 34.5000
```

Στα σημεία όπου σημειώνεται μείωση του ρυθμού μάθησης, αυτό συμβαίνει λόγω αύξησης των απωλειών για αριθμό εποχών ίσο με “wait”, όπως για παράδειγμα:

```
Epoch 2/100: Train Loss = 2.3137, Test Loss = 2.3118, Train Accuracy = 15.6300, Test Accuracy = 15.9800
Epoch 3/100: Train Loss = 3.6492, Test Loss = 3.6486, Train Accuracy = 14.0400, Test Accuracy = 13.8400
Epoch 4/100: Reducing learning rate to 0.000700
Epoch 4/100: Train Loss = 5.5426, Test Loss = 5.5538, Train Accuracy = 21.7360, Test Accuracy = 22.6600
Epoch 5/100: Train Loss = 6.4761, Test Loss = 6.4938, Train Accuracy = 10.1720, Test Accuracy = 10.3000
Epoch 6/100: Reducing learning rate to 0.000490
Epoch 6/100: Train Loss = 5.8247, Test Loss = 5.8523, Train Accuracy = 16.2320, Test Accuracy = 16.7600
Epoch 7/100: Train Loss = 5.7508, Test Loss = 5.7855, Train Accuracy = 20.7340, Test Accuracy = 20.9000
Epoch 8/100: Reducing learning rate to 0.000343
Epoch 8/100: Train Loss = 5.0994, Test Loss = 5.1156, Train Accuracy = 23.0020, Test Accuracy = 22.7900
Epoch 9/100: Train Loss = 4.0553, Test Loss = 4.0698, Train Accuracy = 20.9540, Test Accuracy = 20.9500
Epoch 10/100: Reducing learning rate to 0.000240
```

- **<u>Αρχικοποίηση βαρών (weights)</u>**
  Πραγματοποιήθηκε η μελέτη και η αξιολόγηση διαφορετικών μεθόδων αρχικοποίησης των βαρών στη διαδικασία εκπαίδευσης του RBFNN, διατηρώντας ίδια τα εξής χαρακτηριστικά για το δίκτυο:
  - PCA με 91% (βλ. *2.4, σελ.5)*
  - **Κ-Μέσων** για την επιλογή κέντρων με “*max_iters = 100*”(βλ. *2.5, σελ.7*)
  - Υπολογισμός μοναδικού “σ” (βλ. *2.12.Α, σελ.14*)
  - Κανονικοποίηση Row-wise Max στις ενεργοποιήσεις του κρυφού στρώματος (βλ. *2.12.Α, σελ.14*)
  - “*epochs*” = 100, αρχικό “*learning_rate” = 0.001*”, “*lr_decay*” = 0.7 και “*wait*” = 2
  - Προσθήκη σταθερού bias (+1)

Δοκιμάστηκαν αρκετές φορές τα εξής:

i. *Uniform Initialization* → $w = U(-0.5, 0.5)$. Παρατηρήθηκαν από νωρίς πολλές αυξομειώσεις μικρής κλίμακας στις τιμές της ακρίβειας, όπως για παράδειγμα φαίνεται στην εικόνα:

```
limit = 0.5
weights = np.random.uniform(-limit, limit, size=(G_train_aug.shape[1], 10))
```

```
Epoch 90/100: Train Loss = 1.8575, Test Loss = 1.8584, Train Accuracy = 34.2360, Test Accuracy = 34.7700
Epoch 91/100: Train Loss = 1.8542, Test Loss = 1.8514, Train Accuracy = 33.6060, Test Accuracy = 34.2800
Epoch 92/100: Train Loss = 1.8574, Test Loss = 1.8584, Train Accuracy = 34.2520, Test Accuracy = 34.7700
Epoch 93/100: Train Loss = 1.8535, Test Loss = 1.8508, Train Accuracy = 33.6240, Test Accuracy = 34.3000
Epoch 94/100: Train Loss = 1.8574, Test Loss = 1.8585, Train Accuracy = 34.2820, Test Accuracy = 34.7800
Epoch 95/100: Train Loss = 1.8530, Test Loss = 1.8503, Train Accuracy = 33.6300, Test Accuracy = 34.2100
Epoch 96/100: Train Loss = 1.8575, Test Loss = 1.8587, Train Accuracy = 34.3100, Test Accuracy = 34.8000
Epoch 97/100: Train Loss = 1.8525, Test Loss = 1.8499, Train Accuracy = 33.6360, Test Accuracy = 34.1400
Epoch 98/100: Train Loss = 1.8576, Test Loss = 1.8589, Train Accuracy = 34.3260, Test Accuracy = 34.8300
Epoch 99/100: Train Loss = 1.8521, Test Loss = 1.8496, Train Accuracy = 33.6500, Test Accuracy = 34.0800
Epoch 100/100: Train Loss = 1.8577, Test Loss = 1.8592, Train Accuracy = 34.3480, Test Accuracy = 34.8100
Συνολικός Χρόνος Εκπαίδευσης: 393.25 δευτερόλεπτα
```

ii. *He Uniform Initialization* → $w = U(-\sqrt{\frac{6}{n_{in}}}, \sqrt{\frac{6}{n_{in}}})$, με n_{in} να είναι ο αριθμός των εισόδων (διαστάσεις των χαρακτηριστικών) του επιπέδου, που στην περίπτωση μας είναι το $G\_train\_aug.shape[1]$.

```
limit = np.sqrt(6 / (G_train_aug.shape[1]))
weights = np.random.uniform(-limit, limit, size=(G_train_aug.shape[1], 10))
```

```
Epoch 100/100: Train Loss = 1.8470, Test Loss = 1.8508, Train Accuracy = 34.2640, Test Accuracy = 34.7300
Συνολικός Χρόνος Εκπαίδευσης: 396.30 δευτερόλεπτα
```

iii. *He Normal Initialization* → $w = N(0, \frac{2}{n_{in}})$

```
weights = np.random.randn(G_train_aug.shape[1], 10) * np.sqrt(2 / G_train_aug.shape[1])
```

```
Epoch 100/100: Train Loss = 1.8372, Test Loss = 1.8438, Train Accuracy = 34.8140, Test Accuracy = 35.0500
Συνολικός Χρόνος Εκπαίδευσης: 399.33 δευτερόλεπτα
```

iv. *Xavier Uniform Initialization* $w = U(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}})$, με $n_{out} = 10$ να είναι ο αριθμός των εξόδων του RBFNN.

```
limit = np.sqrt(6 / (G_train_aug.shape[1] + 10))
weights = np.random.uniform(-limit, limit, size=(G_train_aug.shape[1], 10))
```

```
Epoch 100/100: Train Loss = 1.8286, Test Loss = 1.8319, Train Accuracy = 35.2220, Test Accuracy = 35.5600
Συνολικός Χρόνος Εκπαίδευσης: 410.98 δευτερόλεπτα
```

v. *Xavier Normal Initialization*→ $w = N(0, \frac{1}{n_{in}+n_{out}})$

```
weights = np.random.randn(G_train_aug.shape[1], 10) * np.sqrt(1 / (G_train_aug.shape[1] + 10))
```

```
Epoch 100/100: Train Loss = 1.8438, Test Loss = 1.8438, Train Accuracy = 34.8600, Test Accuracy = 34.8500
Συνολικός Χρόνος Εκπαίδευσης: 433.80 δευτερόλεπτα
```

Συνοψίζοντας, συμπεραίνουμε ότι:

- Τα αποτελέσματα απόδοσης ήταν αρκετά **παρόμοια** για όλες τις μεθόδους αρχικοποίησης.
- Δεν παρατηρήθηκε ουσιαστική διαφορά στην ακρίβεια ή στις απώλειες ανάμεσα στις μεθόδους, γεγονός που υποδηλώνει ότι η επιλογή της αρχικοποίησης έχει μικρότερη επίδραση από άλλες παραμέτρους του μοντέλου.
- Οι διακυμάνσεις ήταν εμφανείς κυρίως στην Uniform Initialization, ενώ οι υπόλοιπες μέθοδοι παρουσίασαν πιο σταθερή συμπεριφορά.

- **Αρχικοποίηση bias**

Η προσθήκη του $bias$ σε ένα νευρωνικό δίκτυο είναι κρίσιμη για την εξασφάλιση της ευελιξίας του μοντέλου, επιτρέποντας τη μετατόπιση της συνάρτησης ενεργοποίησης ώστε να μπορεί να προσαρμοστεί καλύτερα στα δεδομένα. Στο πλαίσιο της παρούσας εργασίας, μελετήθηκαν διαφορετικές τιμές αρχικοποίησης του $bias$, όπως φαίνεται παρακάτω:

Αρχικοποίηση bias	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
Σταθερό (+2)	384.83 sec	28.89 %	29.32 %	4.3509	4.3403
Σταθερό (+1)	413.85 sec	35.32 %	35.64 %	1.8276	1.8355
Σταθερό (+0.5)	427.62 sec	34.98 %	35.34 %	1.8290	1.8307
Σταθερό (+0.1)	406.17 sec	35.16 %	35.15 %	1.8295	1.8431

Συνοψίζοντας, συμπεραίνουμε ότι:

- Το $\boldsymbol{bias}$ $(+\mathbf{2})$ έχει τον χαμηλότερο χρόνο εκπαίδευσης και πιθανόν να σχετίζεται με την μεγαλύτερη ευκολία του μοντέλου να προσαρμοστεί στη συγκεκριμένη αρχικοποίηση. Οι χαμηλές τιμές στην ακρίβεια δείχνουν ότι επηρεάζει αρνητικά τη δυνατότητα του δικτύου να διαχωρίσει τις κλάσεις, ενδεχομένως προκαλώντας υπερβολική μετατόπιση στις ενεργοποιήσεις των νευρώνων. Οι απώλειες είναι υψηλές, δείχνοντας περιορισμένη προσαρμογή του μοντέλου.
- Το $\boldsymbol{bias}$ $(+\mathbf{1})$ παρουσιάζει την καλύτερη απόδοση, υποδεικνύοντας ότι επιτρέπει στο δίκτυο να ισορροπήσει τις ενεργοποιήσεις, διατηρώντας καλή προσαρμογή στα δεδομένα. Οι απώλειες είναι οι χαμηλότερες από όλες τις περιπτώσεις, επιβεβαιώνοντας τη σταθερή μάθηση.
- Με τα $\boldsymbol{bias}$ $(+\mathbf{0.1})$ και $(+\mathbf{0.5})$, παρόλο που η **ακρίβεια** είναι παρόμοια με το $bias$ $(+1)$, παρατηρούνται **έντονες αυξομειώσεις** σε αυτήν ήδη από την 6η και 9η εποχή αντίστοιχα, λόγω της ανεπαρκούς αρχικής μετατόπισης των

ενεργοποιήσεων, γεγονός που υποδεικνύει ότι μπορεί να μην παρέχουν την απαιτούμενη σταθερότητα στη σύγκλιση (μάθηση) του μοντέλου.

- Το $\boldsymbol{bias}$ $(+\mathbf{1})$ είναι η πιο αξιόπιστη επιλογή. Προσφέρει την καλύτερη ακρίβεια και τις χαμηλότερες απώλειες, χωρίς αστάθειες κατά τη διαδικασία εκπαίδευσης.

- **<u>Κανονικοποίηση των ενεργοποιήσεων του κρυφού στρώματος</u>**

Η κανονικοποίηση των ενεργοποιήσεων (G) του κρυφού στρώματος (βλ. *2.12.Α, σελ.14*)) αποτελεί μία τεχνική που αποσκοπεί στη σταθεροποίηση της εκπαίδευσης του νευρωνικού δικτύου, περιορίζοντας τις αποκλίσεις στις τιμές των ενεργοποιήσεων. Με αυτόν τον τρόπο επιτυγχάνεται βελτίωση της σταθερότητας και της ακρίβειας, μείωση των απωλειών, καθώς και επιτάχυνση της εκπαίδευσης. Για παράδειγμα με χαρακτηριστικά δικτύου:

- PCA με 91% (βλ. *2.4, σελ.5)*
- **Κ-Μέσων** για την επιλογή κέντρων με "*max_iters = 100*" (βλ. *2.5, σελ.7*)
- Υπολογισμός μοναδικού "σ" (βλ. *2.12.Α, σελ.14*)
- "*epochs*" = 100, αρχικό "*learning_rate" = 0.001*", "*lr_decay*" = 0.7 και "*wait*" = 2
- Προσθήκη σταθερού bias (+1)

Κανον/ποίηση των G	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
Χωρίς	433.13 sec	19.15 %	19.21 %	2.2723	2.2721
Row-wise Max	413.85 sec	35.32 %	35.64 %	1.8276	1.8355
Min-Max	399.19 sec	34.72 %	34.68 %	1.8400	1.8421
Z-Score	400.11 sec	33.98 %	34.67 %	2.1798	2.1836

i. Row-wise Max Normalization

```
# Normalization στο G
G_train /= np.max(G_train, axis=1, keepdims=True)
G_test /= np.max(G_test, axis=1, keepdims=True)
```

```
G_train πριν την κανονικοποίηση: min=3.7284153313769036e-62, max=1.0, mean=0.0011610487667294636
G_test πριν την κανονικοποίηση: min=6.702104437367101e-62, max=0.6964124564527928, mean=0.0011628886561409928
G_train μετά την κανονικοποίηση: min=8.948795388665837e-60, max=1.0, mean=0.033246293020228454
G_test μετά την κανονικοποίηση: min=3.172922632839031e-59, max=1.0, mean=0.034320669826215774
```

ii. Min-Max Scaling

```
# Κανονικοποίηση στο G
G_train = (G_train - np.min(G_train, axis=1, keepdims=True)) / (np.max(G_train, axis=1, keepdims=True) - np.min(G_train, axis=1, keepdims=True) + 1e-9)
G_test = (G_test - np.min(G_test, axis=1, keepdims=True)) / (np.max(G_test, axis=1, keepdims=True) - np.min(G_test, axis=1, keepdims=True) + 1e-9)
```

```
G_train πριν την κανονικοποίηση: min=6.053226403914751e-63, max=1.0, mean=0.0011227717652317402
G_test πριν την κανονικοποίηση: min=5.86710716961337e-63, max=0.6413978858751778, mean=0.0011248074037546476
G_train μετά την κανονικοποίηση: min=0.0, max=0.999999999, mean=0.03299610129707046
G_test μετά την κανονικοποίηση: min=0.0, max=0.9999999984409054, mean=0.03415316614105539
```

iii. Z-Score Normalization

```
# Normalization στο G
G_train = (G_train - np.mean(G_train, axis=1, keepdims=True)) / (np.std(G_train, axis=1, keepdims=True) + 1e-9)
G_test = (G_test - np.mean(G_test, axis=1, keepdims=True)) / (np.std(G_test, axis=1, keepdims=True) + 1e-9)
```

```
G_train πριν την κανονικοποίηση: min=1.4417162462065147e-63, max=0.794577167049343, mean=0.0011839708344211614
G_test πριν την κανονικοποίηση: min=1.483685761006031e-63, max=0.6888762289079498, mean=0.0011846272438980562
G_train μετά την κανονικοποίηση: min=-0.5100856085895478, max=9.949868840360729, mean=7.812417379682302e-19
G_test μετά την κανονικοποίηση: min=-0.4910305091846838, max=9.949865929985867, mean=-4.6629367034256577e-20
```

Συνοψίζοντας, συμπεραίνουμε ότι:

- Η κανονικοποίηση **Row-wise Max** (με βάση τη μέγιστη τιμή σε κάθε γραμμή) οδηγεί στη βέλτιστη ακρίβεια και στις χαμηλότερες απώλειες. Πρόκειται για τη βέλτιστη προσέγγιση σε αυτά τα πειράματα.
- Η **Min-Max** αν και φαίνεται συγκρίσιμη με την προηγούμενη μέθοδο σε ακρίβεια (χαμηλότερη) και απώλειες (υψηλότερες), παρατηρήθηκε ότι η ακρίβεια στο test set συμεχώς μειώνεται μετά την 80η εποχή (34.93%).
- Η κανονικοποίηση με **Z-Score** επιτυγχάνει παρόμοια απόδοση με το Min-Max σε ακρίβεια, αλλά συνοδεύεται από αρκετά αυξημένες απώλειες. Αυτό μπορεί να υποδηλώνει ότι ο τρόπος κεντραρίσματος και τυποποίησης των δεδομένων δεν είναι ιδανικός για τη συγκεκριμένη αρχιτεκτονική.

3.5. Β-Υλοποίηση με χρήση Τυχαίας επιλογής κέντρων

- **Όρος κανονικοποίησης της Least Squares**
  Στη συνέχεια εξετάστηκε η επίδραση διαφορετικών τιμών της υπερπαραμέτρου "$lambda\_reg$" στη διαδικασία εκπαίδευσης του μοντέλου. Αυτή λειτουργεί ως όρος κανονικοποίησης στη συνάρτηση κόστους, συμβάλλοντας στη μείωση της υπερπροσαρμογής και στη βελτίωση της γενίκευσης του μοντέλου.

Χαρακτηριστικά δικτύου:

- PCA με 91% (βλ. *2.4, σελ.5)*
- Υπολογισμός "σ" για κάθε κέντρο (βλ. *2.12.Β, σελ.18*)
- Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

i. Για παράδειγμα με αριθμό κρυφών νευρώνων ίσο με 1000 αν και δεν είναι τόσο εμφανής η λειτουργία που προαναφέρθηκε:

"lambda_reg"	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
0	58.97 sec	52.87 %	50.09 %	2.0797	2.0910
1e-4	66.79 sec	52.39 %	49.90 %	2.0805	2.0915
1e-3	55.53 sec	52.40 %	49.40 %	2.0831	2.0936
1e-2	58.73 sec	51.44 %	49.25 %	2.0947	2.1028
1e-1	54.91 sec	48.83 %	47.53 %	2.1165	2.1199

- Η επιλογή "$lambda\_reg = 0$" προσφέρει την υψηλότερη ακρίβεια εκπαίδευσης και δοκιμής, αλλά ενδέχεται να οδηγεί σε υπερπροσαρμογή.
- Μικρές τιμές κανονικοποίησης $(1e-3, 1e-4)$ δεν σημειώνουν μεγάλη βελτίωση όσον αφορά την απόδοση και τη γενίκευση.
- Η επιλογή "$lambda\_reg = 1e-2$" φαίνεται να εξισορροπεί καλύτερα την απόδοση και τη γενίκευση, με μικρή μόνο πτώση στην ακρίβεια του test set.
- Η επιλογή "$lambda\_reg = 1e-1$" περιορίζει την απόδοση του μοντέλου, καθιστώντας το λιγότερο αποτελεσματικό στη μάθηση από τα δεδομένα.

ii. Ωστόσο, με αριθμό κρυφών νευρώνων ίσο με 7000, ο όρος κανονικοποίησης εμφανέστατα συμβάλει στη μείωση της υπερπροσαρμογής και στη βελτίωση της γενίκευσης του μοντέλου (αποφυγή υπερεκπαίδευσης):

"lambda_reg"	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
0	690.75 sec	71.99 %	54.31 %	1.9387	2.0159
1e-3	711.04 sec	64.71 %	54.62 %	2.0017	2.0402
1e-2	824.82 sec	58.90 %	53.14 %	2.0434	2.0638

- **<u>Κανονικοποίηση των ενεργοποιήσεων του κρυφού στρώματος</u>**

Δεν είναι τόσο απαραίτητη όπως στην Α-Υλοποίηση καθώς οι τιμές των ενεργοποιήσεων ούτως ή άλλως δεν παρουσιάζουν μεγάλες αποκλίσεις ,όπως φαίνεται παρακάτω:

```
G_train πριν την κανονικοποίηση: min=0.0560423871490245, max=1.0, mean=0.6127476904867296
G_test πριν την κανονικοποίηση: min=0.0562102065111622, max=0.9913946130086277, mean=0.6132630927336905
G_train μετά την κανονικοποίηση: min=0.061911558211065094, max=1.0, mean=0.7103536831892945
G_test μετά την κανονικοποίηση: min=0.062215483295661, max=1.0, mean=0.7128907083706486
```

Χαρακτηριστικά δικτύου:

- PCA με 91% (βλ. *2.4, σελ.5)*
- Υπολογισμός "σ" για κάθε κέντρο (βλ. *2.12.Β, σελ.18*)
- Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

Για παράδειγμα με 1000 νευρώνες:

Κανον/ποίηση	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
Ναι	56.35 sec	52.54 %	49.80 %	2.0805	2.0918
Όχι	53.21 sec	52.91 %	50.19 %	2.0780	2.0891

Για παράδειγμα με 7000 νευρώνες:

Κανον/ποίηση	Χρόνος εκπαίδευσης	Training Accuracy	Testing Accuracy	Train Loss	Test Loss
Ναι	824.82 sec	58.49 %	52.93 %	2.0457	2.0682
Όχι	707.26 sec	58.90 %	53.14 %	2.0434	2.0638

4. Σύγκριση του RBFNN με άλλους κατηγοριοποιητές

Σε αυτήν την ενότητα, θα συγκριθεί η απόδοση του **RBFNN** της παρούσας εργασίας σε σχέση με την κατηγοριοποίηση 1 και 3 πλησιέστερου γείτονα (**Nearest Neighbor**) και πλησιέστερου κέντρου κλάσης (**Nearest Class Centroid**). Για την αξιολόγηση των κατηγοριοποιητών πέρα του RBFNN, θα χρησιμοποιηθούν τα αποτελέσματα από την ενδιάμεση εργασία. Αναλυτικότερα:

- **1-NN**, **3-NN**, **NCC** από την ενδιάμεση εργασία με την προσωπική υλοποίηση

- **Α-Υλοποίηση** με χαρακτηριστικά δικτύου:
  - PCA με 91% (βλ. *2.4, σελ.5)*
  - **Κ-Μέσων** για την επιλογή κέντρων με "*max_iters = 100*"(βλ. *2.5, σελ.7*)
  - Υπολογισμός μοναδικού "σ" (βλ. *2.12.Α, σελ.14*)
  - Κανονικοποίηση Row-wise Max στις ενεργοποιήσεις του κρυφού στρώματος (βλ. *2.12.Α, σελ.14*)
  - Εκπαίδευση εξωτερικού στρώματος με **backpropagation**
  - Αριθμός κρυφών νευρώνων ίσος με 100
  - "*epochs*" = 100, αρχικό "*learning_rate" = 0.001*", "*lr_decay*" = 0.7 και "*wait*" = 2
  - Xavier Uniform αρχικοποίηση για τα βάρη
  - Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

- **Β-Υλοποίηση** με χαρακτηριστικά δικτύου:
  - PCA με 91% (βλ. *2.4, σελ.5)*
  - **Τυχαία επιλογή κέντρων** (βλ. *2.5, σελ.8*)
  - Εκπαίδευση εξωτερικού στρώματος με **Least Squares** με **regularization**
  - Αριθμός κρυφών νευρώνων ίσος με 5000
  - "*lamda_reg*" = 1e-2 (βλ. *2.12.Β, σελ.17*)
  - Υπολογισμός "σ" για κάθε κέντρο (βλ. *2.12.Β, σελ.18*)
  - Δυναμική μάθηση του bias κατά την εκπαίδευση (αρχικοποίηση με +1)

Κατηγοριοποιητής	**Χρόνος Εκπαίδευσης**	**Training Accuracy**	**Testing Accuracy**
1-NN	~9566 sec	100%	35%
3-NN	~9695 sec	~95%	33%
NCC	~4.8 sec	~50%	28%
Α-Υλοποίηση	413.85 sec	35.32 %	35.64 %
Β-Υλοποίηση	483.20 sec	57.68 %	52.51%

Όσον αφορά τον χρόνο εκπαίδευσης, συμπεραίνουμε ότι:

- Οι κατηγοριοποιητές NN είναι εξαιρετικά αργοί, μεπολύ υψηλούς χρόνους εκπαίδευσης. Ο υψηλός υπολογιστικός φόρτος οφείλεται στην ανάγκη αποθήκευσης και σύγκρισης με όλα τα δείγματα.
- Ο NCC είναι ο ταχύτερος, καθώς υπολογίζει μόνο τα κέντρα των κλάσεων. Ωστόσο, η απλοϊκή του προσέγγιση περιορίζει σημαντικά την απόδοση.
- Η Α-Υλοποίηση RBF παρουσιάζει σημαντική μείωση του χρόνου εκπαίδευσης, χάρη στη χρήση PCA, K-Μέσων για επιλογή κέντρων, και την πιο αποδοτική διαδικασία εκπαίδευσης με backpropagation.
- Η Β-Υλοποίηση RBF αν και λίγο βραδύτερη επωφελείται από τη χρήση Least Squares και την τυχαία επιλογή κέντρων, προσφέροντας καλύτερη απόδοση χωρίς μεγάλη αύξηση του χρόνου.

Όσον αφορά την ακρίβεια, συμπεραίνουμε ότι:

- Οι κατηγοριοποιητές NN υποφέρουν από υπερπροσαρμογή με πολύ χαμηλή ακρίβεια στο test set.
- Ο NCC παρουσιάζει την χαμηλότερη ακρίβεια, αντανακλώντας την περιορισμένη του ικανότητα για σύνθετα προβλήματα όπως το CIFAR-10.
- Η Α-Υλοποίηση RBF διατηρεί ισορροπία μεταξύ εκπαίδευσης και ελέγχου (αποφυγή υπερεκπαίδευσης), ξεπερνώντας τις παραδοσιακές μεθόδους σε σταθερότητα, αλλά όχι σε απόδοση.
- Η Β-Υλοποίηση RBF υπερέχει σημαντικά όλων, με το υψηλότερο ποσοστά σε ακρίβεια στο test set. Δεν δείχνει να υπερπροσαρμόζεται και οδηγεί σε υψηλότερη γενίκευση, παρά το αυξημένο υπολογιστικό κόστος.