3η ΕΡΓΑΣΙΑ ΣΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

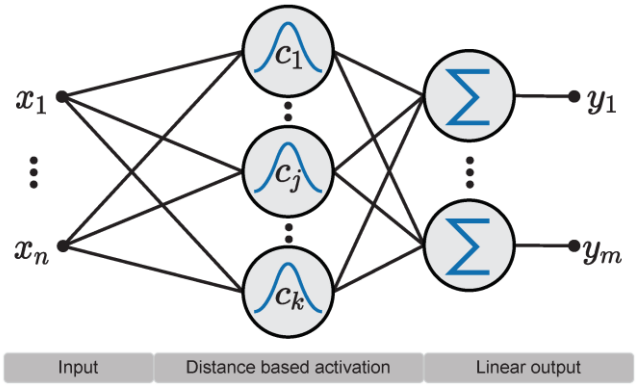
Εργασία του Ανδρέα Σεγκάνι ΑΕΜ 10770

Στη συγκεκριμένη εργασία πραγματοποιήθηκε η ανάλυση ενός αλγορίθμου ταξινόμησης εικόνας σε κλάσης χρησιμοποιώντας τον αλγόριθμο των δικτύων ακτινικής βάσης. Στην εργασία πραγματοποιήθηκαν διάφορα πειράματα με διάφορες παραμέτρους και αναγράφονται όλα τα αποτελέσματα των training και testing accuracies ανά εποχή. Αξίζει να σημειωθεί ότι ο μέσος χρόνος εκτέλεσης του συνολικού προγράμματος είναι 30 λεπτά με μικρές διαφορές , ανάλογα τις συναρτήσεις που θα επιλεγούν. Η μοναδική φορά που είχαμε διπλάσιο + χρόνο ήταν όταν επιλέχθηκαν παραπάνω κέντρα.

I. ΕΙΣΑΓΩΓΗ

Τα δίκτυα ακτινικής βάσης (RBF) είναι ένα δίκτυο εμπρόσθιας τροφοδότησης με τους νευρώνες του κρυφού επιπέδου να διαχειρίζονται μια ακτινική, μη γραμμική συνάρτηση βάσης.

Η Δομή ενός RBF φαίνεται παρακάτω:



This paragraph of the first footnote will contain the date on which you submitted your paper for review, which is populated by IEEE. It is IEEE style to display support information, including sponsor and financial support acknowledgment, here and not in an acknowledgment section at the end of the article. For example, “This work was supported in part by the U.S. Department of Commerce under Grant 123456.” The name of the corresponding author appears after the financial information, e.g. *(Corresponding author: Second B. Author).* Here you may also indicate if authors contributed equally or if there are co-first authors.

The next few paragraphs should contain the authors’ current affiliations, including current address and e-mail. For example, First A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (e-mail: author@ boulder.nist.gov).

Second B. Author Jr. was with Rice University, Houston, TX 77005 USA. He is now with the Department of Physics, Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar.colostate.edu).

Third C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA, on leave from the National Research Institute for Metals, Tsukuba 305-0047, Japan (e-mail: author@nrim.go.jp).

Mentions of supplemental materials and animal/human rights statements can be included here.

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org

Τα RBF δίκτυα μπορούν να χρησιμοποιηθούν για μάθηση συναρτήσεων και ταξινόμηση δεδομένων με τον ίδιο τρόπο όπως και στα δίκτυα MLP. Όπως φαίνεται και στη παραπάνω εικόνα, η δομή τους έχει ομοιότητες με εκείνη των απλών δικτύων feedforward ,αφού και οι δύο αποτελούνται από ένα επίπεδο εισόδου, ένα κρυφό επίπεδο και από ένα επίπεδο εξόδου.

Ο τρόπος λειτουργίας του RBF είναι ο εξής. Αρχικά το δίκτυο δέχεται τα σήματα εισόδου , αφού έχουν υποστεί κανονικοποίηση , και έπειτα διαδίδονται στο κρυφό επίπεδο. Στο κρυφό επίπεδο , τα σήματα επεξεργάζονται από τους νευρώνες που θέτουμε. Ο αριθμός των νευρώνων είναι ίσος με τον αριθμό των κέντρων που επιλέγουμε. Ένας νευρώνας περιέχει μια ακτινική συνάρτηση βάσης φ , από την οποία περνάει το σήμα για να βγει στην έξοδο. Επιπλέον, κάθε κέντρο έχει ίδια διάσταση με το σήμα εισόδου και έχουν διάσταση ίση με τον αριθμό μεταβλητών εισόδου, δηλαδή για το cifar10, 32x32x3 = 3072. Τέλος , αφού τα δεδομένα επεξεργαστούν στο κρυφό επίπεδο , περνάνε στην έξοδο του δικτύου με μια γραμμική σχέση.

Στο δίκτυο που κατασκευάστηκε αξιοποιήθηκαν τα εξής στοιχεία:

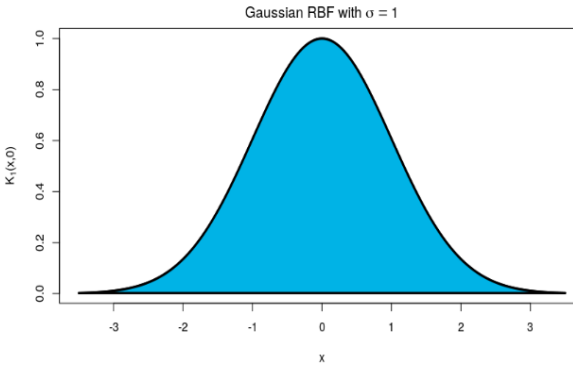
- Κάθε νευρώνας του κρυφού επιπέδου έχει ένα κέντρο c, που αντιπροσωπεύει ένα σημείο στον χώρο των εισόδων.
- Υπολογισμός της Ευκλείδιας απόστασης ενός κέντρου από κάποιο παράδειγμα προς εκπαίδευση. Δηλαδή υπολογίζεται το r = ||x-c||
- Συνάρτηση φ : Η ακτινική συνάρτηση βάσης μετατρέπει τον μη γραμμικά διαχωρίσιμο χώρο εισόδου σε έναν μη γραμμικό χώρο χαρακτηριστικών. Εκεί τα δεδομένα μπορούν να διαχωριστούν γραμμικά στο επίπεδο εξόδου. Η έξοδος του κρυφού νευρώνα υπολογίζεται ως φ(r) για κάποιο r_j .
- Η έξοδος ενός δικτύου RBF για ένα παράδειγμα προς εκπαίδευση i είναι :
  $y_i(x_i) = \sum_{j=1}^{N} w_j * \varphi_j(x_j, c_j)$

II. ΤΥΠΟΙ ΣΥΝΑΡΤΗΣΕΩΝ ΒΑΣΗΣ

Στα RBF οι συναρτησεις βάσης καθορίζουν πως υπολογίζεται η απόκριση ενός κρυφού νευρώνα με βάση την απόσταση του εισερχόμενου δεδομένου από το κέντρο του. Υπάρχουν διάφοροι τύποι συναρτήσεων βάσης, κάθε μία με διαφορετικά χαρακτηριστικά.

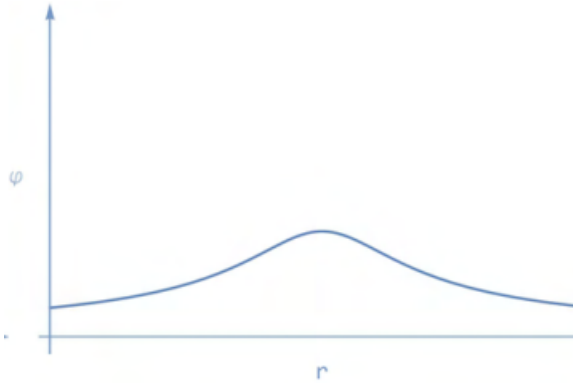
1. Gaussian RBF : Στη Γκαουσιανή συνάρτηση

βάσης ισχύει ότι φ(r) = $e^{\frac{r^2}{2\sigma^2}}$, όπου r η ευκλείδια απόσταση του ενός συγκεκριμένου παραδείγματος προς εκπαίδευση από το κέντρο cj .



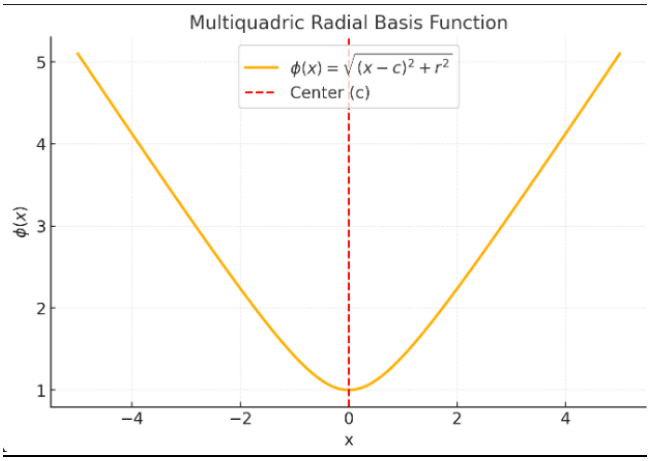
Στη συγκεκριμένη εικόνα βλέπουμε ότι έχουμε τυπική απόκλιση ίση με 1 . Εάν το δεδομένο x είναι πολύ κοντά στο κέντρο του νευρώνα τότε η γκαουσιανή συνάρτηση πλησιάζει στο μέγιστο, και έτσι έχουμε ισχυρότερο activation , το οποίο χρησιμοποιείται στον υπολογισμό των βαρών. Επομένως από τη γραφική βλέπουμε πως καθώς απομακρυνόμαστε από το κέντρο, τότε φ->0

2. Inverse multiquadric RBF : Στην ανάστροφη πολυτετραγωνική συνάρτηση έχουμε: φ(r)=$1/(r^2 + c^2)^{1/2}$.



Παρατηρούμε ότι έχουμε μια πιο ομαλή και σταδιακή απόσβεση το οποίο είναι κατάλληλο για κοντινά δεδομένα για τη βελτίωση της γενίκευσης.

3. Multiquadrics : φ(r)=$1/(r^2 + c^2)^{1/2}$



Εδώ παρατηρούμε ότι καθώς αποκρινόμαστε από το κέντρο η τιμή της φ αυξάνεται. Συνήθως χρησιμοποιείται , σε περιπτώσεις όπου η αυξανόμενη απόκριση είναι επιθυμητή και επιτρέπει μεγαλύτερη ευελιξία στη μοντελοποίηση δεδόμενων.

ΙΙΙ. ΑΛΓΟΡΙΘΜΟΣ

```python
def rbf(x, c, s):

    #INVERSE
    #return 1.0 / np.sqrt(np.linalg.norm(x - c)**2 + s**2)

    #MULTIQUADRICS
     # return np.sqrt(np.linalg.norm(x - c)**2 + s**2)

    #GAUSSIAN
    return np.exp(-np.linalg.norm(x - c)**2 / (2 * s**2))
```

Αρχικά ορίζουμε τους διαφορετικούς τύπους συναρτήσεων και κάθε φορά που θέλω να επιλέξω έναν συγκεκριμένο τύπο βάζω σχόλιο στα υπόλοιπα.

```python
def train(self, X_train, y_train, centers, sigma):
    #Εκπαίδευση του δικτύου RBF
    self.centers = centers
    self.sigma = sigma
    self.weights = np.random.randn(self.num_centers, self.num_classes)

    for epoch in range(self.epochs):
        predictions = np.zeros((X_train.shape[0], self.num_classes))
        for i in range(X_train.shape[0]):
            activations = np.array([rbf(X_train[i], c, sigma) for c in centers])
            predictions[i] = activations.dot(self.weights)

            #Ενημέρωση βαρών (κανόνας δέλτα)
            error = y_train[i] - predictions[i]  #Σφάλμα:error = y_true - y_pred
            self.weights += self.lr * np.outer(activations, error)  #w = w + lr * activation * error
```

Στη συνέχεια του αλγόριθμου μας υλοποιείται η διαδικασία εκπαίδευσης του RBF με τον ορισμό της συνάρτησης train. Στην αρχή αρχικοποιούνται τα βάρη με τυχαίο τρόπο. Μέσα στο for loop αρχικοποιούνται και οι προβλέψεις για κάθε δείγμα x_train που έχουμε σε κάθε εποχή. Το rbf(X_train[i], c, sigma) υπολογίζει την έξοδο ενός συγκεκριμένου νευρώνα στο κρυφό επίπεδο, όμως με την εισαγωγή for c in centers , περνάμε το x_train από όλους τους νευρώνες του κρυφού επιπέδου. Το αποτέλεσμα είναι ένας πίνακας με όνομα

activations που ουσιαστικά περιέχει τις ενεργοποιήσεις όλων των νευρώνων για το συγκεκριμένο δεδομένο στο X(i)

Επιπλέον, έχουμε ενημέρωση των βαρών με τον κανόνα Δέλτα. Πιο συγκεκριμένα για τον κανόνα δέλτα ισχύει ότι:

$$\mathbf{w}(k+1) = \mathbf{w}(k) + \beta(k)\,(d(k) - y(k))\,\mathbf{a}(k)$$

Όπου β(k) το learning rate , το a(k) η συνάρτηση ενεργοποίησης και (d(k)-y(k)) η διαφορά μεταξύ της επιθυμητής εξόδου και της πρόβλεψης του δικτύου (error).
Αρα τα weights ενημερώνονται με εξωτερικό γινόμενο ,αφού θέλουμε πίνακα, σε w=w+lr*error*φ(x). Τα στοιχεία του πίνακα των βαρών αφορούν κάθε βάρος για κάθε έναν νευρώνα.

```
def compute_sigma(centres):

    num_centres = centres.shape[0]
    dists = np.linalg.norm(centres[:, None] - centres[None, :], axis=2)
    mean_dist = np.mean(dists[np.triu_indices(num_centres, k=1)])

    sigma = mean_dist / np.sqrt(2 * num_centres)
    return sigma
```

Αυτό το μέρος του κώδικα υπολογίζει τη παράμετρο σ, δηλαδή τη τυπική απόκλιση που χρησιμοποιείται στο rbf. Το σ καθορίζεται από τις αποστάσεις μεταξύ των κέντρων. Τα πλάτη θέλουμε να είναι ίσα μεταξύ τους για μια σταθερή συνάρτηση βάσης , σύμφωνα με τον τύπο σ = d/sqrt(2p) , όπου d οι αποστάσεις των κέντρων και p ο αριθμός των νευρώνων.

dists = np.linalg.norm(centres[:, None] - centres[None, :], axis=2). Εδώ υπολογίζονται όλες οι αποστάσεις μεταξύ των κέντρων. Ο πίνακας μπορεί να επεκτείνεται συνεχώς. Στη συνέχεια υπολογίζονται οι αποστάσεις μεταξύ διαφορετικών κέντρων και εξάγονται τα στοιχεία και υπολογίζεται η μέση τιμή αυτών για να χρησιμοποιηθεί στον τύπο. Αυτή διαιρείται με τη ρίζα του διπλάσιου αριθμού του αριθμού των κέντρων ,όπως μας λέει η θεωρία.

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5]),  # Κανονικοποίηση
    transforms.Lambda(lambda x: x.view(-1))  # Flatten εικόνων
])

train_dataset = datasets.CIFAR10(root='./data', train=True, download=True, transform=transf
test_dataset = datasets.CIFAR10(root='./data', train=False, download=True, transform=transf

# Μετατροπή σε numpy arrays
X_train = np.array([train_dataset[i][0].numpy() for i in range(len(train_dataset))])
y_train = np.array([train_dataset[i][1] for i in range(len(train_dataset))])
X_test = np.array([test_dataset[i][0].numpy() for i in range(len(test_dataset))])
y_test = np.array([test_dataset[i][1] for i in range(len(test_dataset))])
```

Ο κώδικας φορτώνει το dataset CIFAR-10 και εφαρμόζει μετασχηματισμούς στις εικόνες, όπως μετατροπή τους σε tensors, κανονικοποίηση στο εύρος [-1, 1], και flattening σε μονοδιάστατα διανύσματα. Στη συνέχεια, οι εικόνες εκπαίδευσης και δοκιμής, μαζί με τις αντίστοιχες ετικέτες, μετατρέπονται από PyTorch tensors σε NumPy arrays, ώστε να είναι έτοιμα για χρήση σε μοντέλα μηχανικής μάθησης εκτός του PyTorch. Το αποτέλεσμα είναι ότι οι εικόνες και οι ετικέτες αποθηκεύονται στους πίνακες X_train, y_train, X_test, και y_test για εκπαίδευση και αξιολόγηση.

```
def select_centers(X, num_centers, method , random_state=None):

    if method == 'kmeans':
        kmeans = KMeans(n_clusters=num_centers, random_state=random_state)
        kmeans.fit(X)
        centers = kmeans.cluster_centers_
    elif method == 'random':
        np.random.seed(random_state)
        centers = X[np.random.choice(X.shape[0], num_centers, replace=False)]
    else:
        raise ValueError("Wrong method. Use 'kmeans' or 'random'.")
    return centers
```

Σε συνέχεια του κώδικα υλοποιείται μια συνάρτηση που επιλέγει τα κέντρα των RBF νευρώνων για ένα RBF δίκτυο, με δύο δυνατές μεθόδους: είτε μέσω του αλγορίθμου k-means είτε τυχαία.

Για τον αλγόριθμο k-means ο αλγόριθμος είναι ο εξής:

a. Αρχικοποίηση των K κέντρων τυχαία.

b. Υπολογίζεται η ευκλείδια απόσταση του xi του από το κάθε κάθε κέντρο μj με την ευκλείδια απόσταση και την αποθηκεύουμε σε ένα πίνακα. Κάθε σημείο κατατάσσεται στο πλησιέστερο κέντρο (cj = argmin(νορμα(xi-mj))).

c. Υπολογίζονται τα νέα κέντρα μj = 1/|Cj| * sum(xi) , όπου Cj το σύνολο των σημείων που ανήκουν στο cluster j και |Cj| το πλήθης των σημείων.

d. Επανάληψη μέχρι να συγκλίνουμε σε έναν μικρό αριθμό τύπου e-4 , όπου εκεί τα κέντρα πλέον δεν αλλάζουν και έχουμε τερματισμό.

```
def k_means(X, k, max_iters=100, tol=1e-8, random_state=None):

    if random_state is not None:
        np.random.seed(random_state)

    #Αρχικοποίηση των κέντρων
    n_samples, n_features = X.shape
    centroids = X[np.random.choice(n_samples, k, replace=False)]

    for iteration in range(max_iters):
        distances = np.linalg.norm(X[:, None] - centroids[None, :], axis=2)

       # κατηγοριοποίηση των xi
        labels = np.argmin(distances, axis=1)

        #Υπολογισμός νέων κέντρων
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)])

        #Έλεγχος για σύγκλιση
        if np.max(np.linalg.norm(new_centroids - centroids, axis=1)) < tol:
            print(f"Σύγκλιση μετά από {iteration + 1} επαναλήψεις.")
            break

        centroids = new_centroids

    return centroids
```

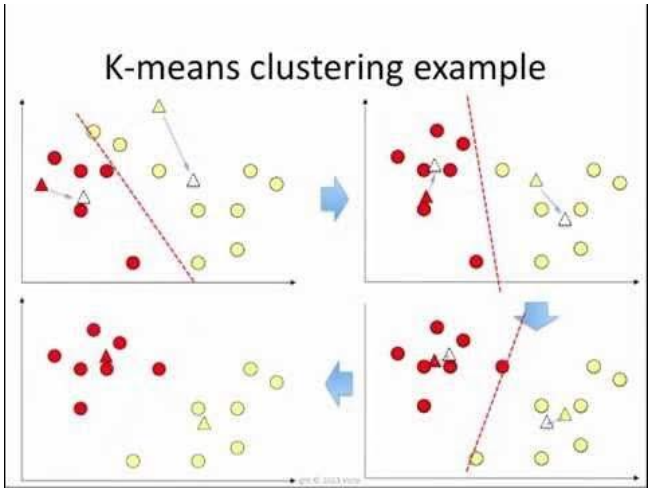
Στον αλγόριθμο μας χρησιμοποιήθηκε η from-scratch συνάρτηση για τον υπολογισμό των k-means ,αλλά χρειάστηκε να εφαρμόσουμε και pca στα δεδομένα μας ώστε να φτάσουν στο 90% της πληροφορίας. Γι' αυτό τον λόγο

εφαρμόσαμε και την έτοιμη συνάρτηση για τον υπολογισμό k-means, πιο συγκεκριμένα, δημιουργείται αντικείμενο KMeans από τη βιβλιοθήκη scikit-learn , και έτσι συγκρίνω τα δύο αποτελέσματα.

Στη μέθοδο της τυχαίας επιλογής , τα κέντρα επιλέγονται τυχαία με τη βοήθεια της np.random , αλλά σε αυτή τη περίπτωση κάθε φορά που τρέχει το πρόγραμμα αναμένουμε και διαφορετικά αποτελέσματα.

```
sigma = compute_sigma(centers)
sigma = 2*sigma

print(f"Computed Sigma: {sigma:.4f}")

# One-Hot Encoding
num_classes = 10
y_train_one_hot = np.eye(num_classes)[y_train]
y_test_one_hot = np.eye(num_classes)[y_test]

# Εκπαίδευση
rbf_net = RBFNetwork(num_centers=num_centers, num_classes=num_classes, learning_rate=0.01, epochs=50)
rbf_net.train(X_train_pca, y_train_one_hot, centers, sigma)

#Αξιολόγηση
y_pred = rbf_net.predict(X_test_pca)
test_accuracy = accuracy_score(y_test, y_pred)  # Ακρίβεια
print(f"Test Accuracy: {test_accuracy:.4f}")
```

Ο κώδικας αρχικά υπολογίζει την τιμή του **σ**, το οποίο καθορίζει το εύρος των RBF συναρτήσεων, με βάση τα κέντρα των RBF νευρώνων. Στη συνέχεια, μετατρέπει τις ετικέτες εξόδου (y) σε μορφή **one-hot encoding** για την αντιμετώπιση της κάθε κατηγορίας ως ανεξάρτητη , χρησιμοποιώντας τον αριθμό κατηγοριών (10 για το CIFAR-10). Δημιουργείται ένα αντικείμενο του RBF δικτύου με προκαθορισμένες παραμέτρους (π.χ., ρυθμός μάθησης, αριθμός εποχών), και το δίκτυο εκπαιδεύεται με τα δεδομένα εκπαίδευσης, τους υπολογισμένους νευρώνες και τη μεταβλητή σ. Τέλος, αξιολογείται η απόδοση του δικτύου στα δεδομένα δοκιμής, μετρώντας την ακρίβεια εκπαίδευσης (train accuracy) μέσω της συνάρτησης accuracy_score και εκτυπώνει το αποτέλεσμα , καθώς και το train accuracy του μοντέλου.

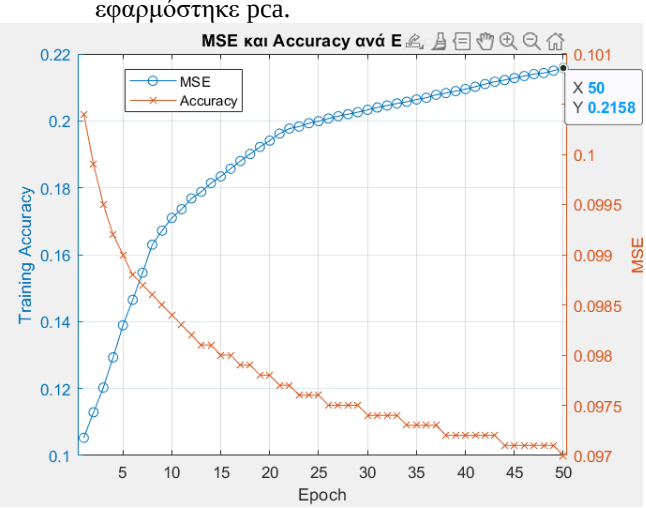


IV. ΑΠΟΤΕΛΕΣΜΑΤΑ

Για τα αποτελέσματα των πειραμάτων έφτιαξα graphs στο Matlab που δείχνουν πως επηρεάζεται το training accuracy ανά εποχή καθώς και αναφέρω και το τελικό training accuracy του κάθε πειράματος.

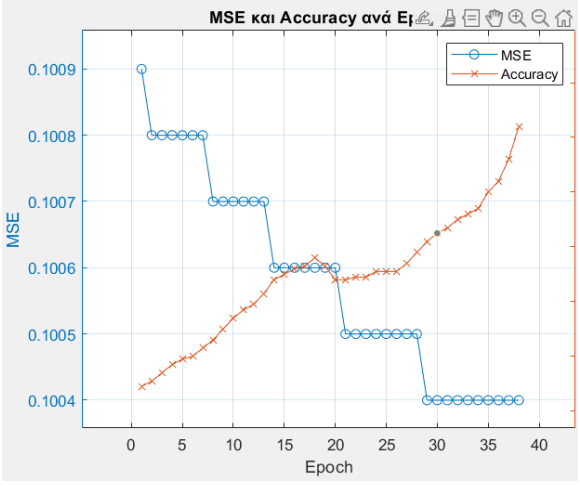
Αρχικά στα πρώτα πειράματα έχω learning rate 0.01 και 50 εποχές και χρησιμοποιώ τη συνάρτηση Gaussian. Επίσης δεν χρησιμοποιώ pca οπότε για τα k means στη περίπτωση αυτή υλοποιείται από έτοιμο κώδικα.

- K-means από έτοιμη βιβλιοθήκη για 50 εποχές και με τιμή σ διπλάσια του εαυτού του (5.184). Δεν εφαρμόστηκε pca.



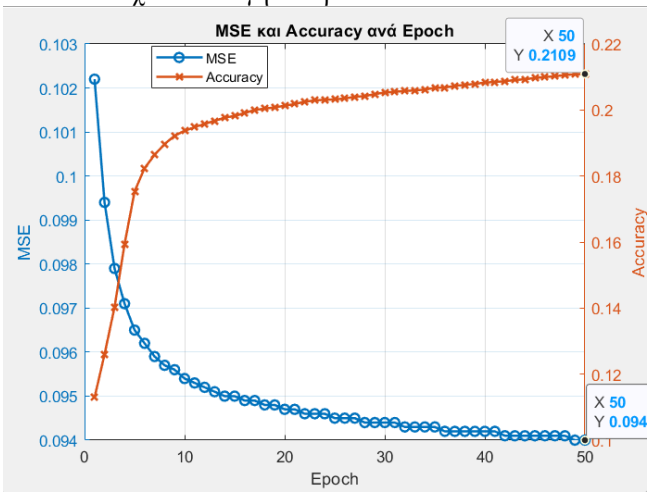
Βλέπουμε ότι το training accuracy θα συγκλίνει περίπου στο 0.23. Το μέσο τετραγωνικό σφάλμα βλέπουμε να πέφτει στο 0.097. Το training accuracy είναι 0.2193 και ο αλγόριθμος έτρεξε για 27 λεπτά συνολικά. Ένας προβληματισμός είναι το ότι το training accuracy και το testing accuracy είναι κοντά αριθμιτικά. Για 70 εποχές το accuracy έφτασε στο 0.2378 και στην 70κοστή έμεινε στο 0.2368, και το training accuracy έφτασε στο 0.2414.

- Τυχαία επιλογή κέντρων με τιμή σ την οποία δεν διπλασιάζω.



Βλέπουμε ότι σχετικά με τη τιμή 2σ το accuracy είναι αρκετά χαμηλό , οπότε σταματώ τη διαδικασία και εκ νέου θα έχω:

- σ=2σ και τυχαία επιλογή κέντρων



Εδώ το training accuracy φτάνει στη τιμή 0.2109 και το mse πέφτει στη τιμή 0.094. Το test accuracy είναι 0.2164 . Παρατηρούμε ότι το k-means μας έδωσε λίγο καλύτερα αποτελέσματα. Παρόλα αυτά έτρεξα ξανά τον αλγόριθμο των τυχαίων κέντρων και μου έδωσε λίγο καλύτερα αποτελέσματα με το training accuracy να φτάνει στο 0.2217 και το test accuracy να φτάνει στο 0.23 , ενώ τη Τρίτη φορά μου έβγαλε αποτελέσματα κάτω της αρχικής προσπάθειας, οπότε θεωρώ πως η τυχαία επιλογή των κέντρων είναι λίγο ασταθής. Ο αλγόριθμος έτρεξε και αυτός γύρω στα 24 λεπτά επειδή λογικά δεν χρειάστηκε να υπολογίσει τα κέντρα.

- Εδώ εφαρμόζω 200 κέντρα και βλέπω το accuracy της περίπτωσης των τυχαίων κέντρων.
  a. Περίπτωση όπου δεν διπλασιάζω το σ:

Epoch 49/50	MSE: 0.0996	Accuracy: 0.2033
Epoch 50/50	MSE: 0.0996	Accuracy: 0.2033

Το training accuracy είναι 0.2071 και το μοντέλο έτρεξε για περίπου μία ώρα

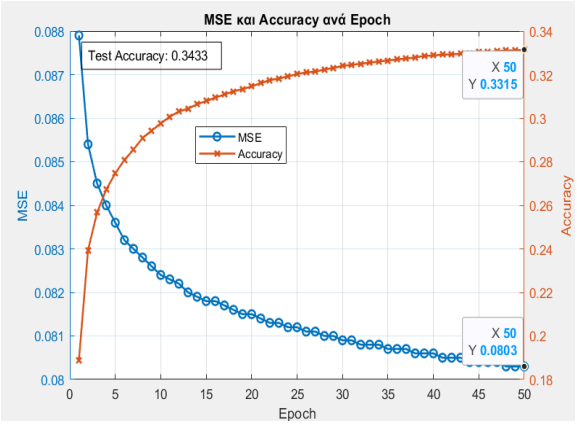
  a. Περίπτωση όπου σ =σ^2

Epoch 49/50	MSE: 0.0824	Accuracy: 0.3286
Epoch 50/50	MSE: 0.0824	Accuracy: 0.3286

Το test accuracy είναι 0.3358, εμφανώς καλύτερα αποτελέσματα από κάθε άλλη μέτρηση

Τώρα θα χρησιμοποιήσω τη συνάρτηση inverse multi και θα έχουμε τα εξής αποτελέσματα

- K-means με 50 κέντρα και σ=2σ με learning rate 0.01 . Ο αλγόριθμος έτρεξε για 27 λεπτά.



Βλέπουμε ότι το training accuracy φτάνει το 0.3315 ενώ το mse φτάνει τη τιμή 0.0803. Το test accuracy έφτασε τη τιμή 0.3433. Βλέπουμε ότι η συνάρτηση inverse είναι πιο αποδοτική από τη gaussian.

- Στη περίπτωση τυχαίων κέντρων και σ=2σ και learning rate 0.01 τα αποτελέσματα είναι αυτά

  Epoch 48/50 - MSE: 0.0834, Accuracy: 0.2841
  Epoch 49/50 - MSE: 0.0833, Accuracy: 0.2846
  Epoch 50/50 - MSE: 0.0833, Accuracy: 0.2849
  Test Accuracy: 0.2928. Συγκριτικά το k -means είναι πιο αποδοτικό
- Στη περίπτωση τυχαίων κέντρων και σ=2σ και learning rate 0.001 τα αποτελέσματα είναι αυτά
  Epoch 48/50 - MSE: 0.0868, Accuracy: 0.2093
  Epoch 49/50 - MSE: 0.0868, Accuracy: 0.2102
  Epoch 50/50 - MSE: 0.0868, Accuracy: 0.2109
  Test Accuracy: 0.2119. Συγκριτικά το k -means είναι πιο αποδοτικό

Στη περίπτωση του multiquadric μου εμφανίζει συνεχώς απόδοση 0.1 και test accuracy 0.1 . Πειράζω τις παραμέτρους σ, pca , τρόπο επιλογής κέντρων και υπολογίζω ξανά αλλά δεν αλλάζει. Το mse φαίνεται να είναι αρκετά μικρό , τύπου NAN , οπότε ψάχνω να δω τι γίνεται με τις αποστάσεις των σημείων από τα κέντρα και βρέθηκε ότι η μέγιστη απόσταση είναι 72.174 και η ελάχιστη 0.0 πράγμα που σημαίνει ότι φτάνω σε nan ή inf τιμές και υπάρχει θέμα στον υπολογισμό της φ. Λόγω των μεγάλων αποστάσεων το multiquadric δεν είναι κατάλληλο.

Εφαρμόζω PCA, ελαττώνουμε τη πληροφορία στο 90% και τότε θα έχω τα εξής αποτελέσματα,

GAUSSIAN

- K-means με σ = 2σ , με δικό μου κώδικα
  Epoch 48/50 - MSE: 0.0955, Accuracy: 0.2315
  Epoch 49/50 - MSE: 0.0955, Accuracy: 0.2320
  Epoch 50/50 - MSE: 0.0955, Accuracy: 0.2324
  Test Accuracy: 0.2405
- K-means χωρίς διπλασιασμό σ , με δικό μου κώδικα
  Epoch 48/50 - MSE: 0.1000, Accuracy: 0.0892
  Epoch 49/50 - MSE: 0.1000, Accuracy: 0.0893
  Epoch 50/50 - MSE: 0.1000, Accuracy: 0.0894
  Test Accuracy: 0.0898
- Τυχαία επιλογή κέντρων με σ=2σ ,έχουμε
  Epoch 48/50 - MSE: 0.0929, Accuracy: 0.2265
  Epoch 49/50 - MSE: 0.0929, Accuracy: 0.2268
  Epoch 50/50 - MSE: 0.0929, Accuracy: 0.2269
  Test Accuracy: 0.2389

Στη συνέχεια βάζω παραπάνω κέντρα(100) και για τυχαία επιλογή κέντρων έχουμε
Epoch 48/50 - MSE: 0.0929, Accuracy: 0.2249
Epoch 49/50 - MSE: 0.0929, Accuracy: 0.2252
Epoch 50/50 - MSE: 0.0929, Accuracy: 0.2253
Test Accuracy: 0.2320

Ενώ με k-means
Epoch 48/50 - MSE: 0.0992, Accuracy: 0.1876
Epoch 49/50 - MSE: 0.0992, Accuracy: 0.1880
Epoch 50/50 - MSE: 0.0992, Accuracy: 0.1883
Test Accuracy: 0.1898

Ενώ με learning rate 0.001 και 100 κέντρα για τυχαία επιλογή έχουμε
Epoch 27/50 - MSE: 0.1019, Accuracy: 0.0955
Epoch 28/50 - MSE: 0.1019, Accuracy: 0.0957
Epoch 29/50 - MSE: 0.1019, Accuracy: 0.0957
Epoch 30/50 - MSE: 0.1019, Accuracy: 0.0959
Εκεί, σταμάτησα τη διαδικασία αφού είναι πολύ χαμηλή η απόδοση (~9%)

Για 50 κεντρα και learning rate 0.1 και σ=2σ (5.129) για k-means έχουμε
Epoch 48/50 - MSE: 0.0945, Accuracy: 0.2567
Epoch 49/50 - MSE: 0.0945, Accuracy: 0.2570
Epoch 50/50 - MSE: 0.0945, Accuracy: 0.2573
Test Accuracy: 0.2591

Για 100 κέντρα και learning rate 0.1 έχουμε

INVERSE MULTIQUADRICS

- K-means με σ =2σ
  Epoch 48/50 - MSE: 0.0826, Accuracy: 0.2925
  Epoch 49/50 - MSE: 0.0825, Accuracy: 0.2931
  Epoch 50/50 - MSE: 0.0825, Accuracy: 0.2933
  Test Accuracy: 0.3035
- Τυχαία επιλογή κέντρων με σ=2σ ,έχουμε
  Epoch 48/50 - MSE: 0.0834, Accuracy: 0.2841
  Epoch 49/50 - MSE: 0.0833, Accuracy: 0.2846
  Epoch 50/50 - MSE: 0.0833, Accuracy: 0.2849
  Test Accuracy: 0.2928

ΣΥΓΚΡΙΣΗ ΜΕ ΑΛΛΕΣ ΜΕΘΟΔΟΥΣ

METHOD	TRAINING	TESTING	TIME
MLP	0.8083	0.4517	30 min (με χρήση cpu)
K-NN (k=3)	0.4	0.3303	20 min
NCC	0.3	0.2274	Κάποια δευτερόλεπτα

ΣΥΜΠΕΡΑΣΜΑ ΚΑΙ ΣΧΟΛΙΑ

Από τα πειράματα που τρέξαμε μπορούμε να καταλάβουμε ότι η inverse έχει γενικά καλύτερα αποτελέσματα από τη gaussian με εμφανώς καλύτερα αποτελέσματα στη training accuracy (average 0.34 vs 0.24) . Η καλύτερη επιλογή για τη gaussian είναι 50 κέντρα με learning rate 0.1 με pca για τη k- means , ενώ για τη περίπτωση των τυχαίων κέντρων είναι όταν σ=σ^2 learning rate 0.01 και χωρις pca και την επιλογή των 50 κέντρων. Για την inverse, η πιο αποτελεσματική επιλογή ήταν η 0.01 lr , 50 κέντρα χωρίς pca με k-means. Βέβαια, από τον πίνακα βλέπουμε ότι πιο αποτελεσματική μέθοδος ταξινόμησης είναι ο mlp και μετά ,ανάλογα με τις παραμέτρους του αλγορίθμου , καλύτερο testing accuracy έχουν rbf , knn και ncc.