ΠΑΝΕΠΙΣΤΗΜΙΟ
ΙΩΑΝΝΙΝΩΝ

ΣΧΟΛΗ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΚΠΑΙΔΕΥΣΗ RBF ΔΙΚΤΥΩΝ ΜΕ ΓΕΝΕΤΙΚΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ

Άγγελος Κώστας

Επιβλέπων: Ιωάννης Τσούλος Αναπληρωτής Καθηγητής

Άρτα, Σεπτέμβριος, 2022

TRAINING RBF NETWORK WITH TWO-PHASE LEARNING USING HYBRID GENETIC ALGORITHM

Εγκρίθηκε από τριμελή εξεταστική επιτροπή

Άρτα, 26/9/22

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής

   Ιωάννης Τσούλος,

   Αναπληρωτής Καθηγητής
2. Μέλος επιτροπής

   Αλέξανδρος Τζάλλας,

   Επίκουρος Καθηγητής
3. Μέλος επιτροπής

   Χαριλόγης Βασίλειος

   Αναπληρωτής Καθηγητής

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Κώστας Άγγελος

Υπογραφή:

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον επιβλέπων καθηγητή μου κύριο Ιωάννη Τσούλο για την πολύτιμη βοήθεια και καθοδήγηση που μου πρόσφερε με χρήσιμες συμβουλές για την ολοκλήρωση της εργασίας. Επίσης θα ήθελα να ευχαριστήσω τους καθηγητές του τμήματος Πληροφορικής και Τηλεπικοινωνιών οι οποίοι συνέβαλαν στην εξέλιξη μου και με τους οποίους είχα άριστη συνεργασία. Τέλος, ευχαριστώ την οικογένεια μου και τους φίλους-συμφοιτητές για την στήριξή τους.

Περίληψη

Η παρούσα εργασία εισάγει τον αναγνώστη στις βασικές έννοιες που αφορούν έναν συγκεκριμένο τύπο τεχνητού νευρωνικού δικτύου, που ονομάζεται Radial Basis Function Network. Αναλύεται η δομή και εξηγείται ο ακριβές τρόπος λειτουργίας του από την τροφοδότηση των εισόδων έως και την παραγωγή της εξόδου. Παράλληλα, αντιμετωπίζει το πρόβλημα ανάπτυξης ενός δικτύου RBF ως ένα θέμα βελτιστοποίησης το οποίο μπορεί να λυθεί με την χρήση γενετικών αλγορίθμων. Ακολουθεί ειδικό κεφάλαιο ανάλυσης των γενετικών αλγορίθμων αλλά και το πως αυτοί μπορούν να παραμετροποιηθούν για να λύσουν το πρόβλημα βελτιστοποίησης του RBF. Τέλος, παρουσιάζονται τμήματα από την υλοποίηση των παραπάνω στην γλώσσα προγραμματισμού C++ αλλά και πειραματικά αποτελέσματα σε γνωστά datasets.

Λέξεις-κλειδιά: RBF network, Νευρωνικό δίκτυο, Βελτιστοποίηση, Γενετικοί αλγόριθμοι

Abstract

This thesis introduces the reader to the basic principles regarding Radial Basis Function Networks. The first chapter provides an overview of the design and architecture while at the same time viewing the issue as an optimization problem that can be solved using a hybrid genetic algorithm. A specific chapter is dedicated showing experimental results between the classical approach of an RBF network versus the specifically designed algorithm for the optimization of the RBF output. To sum up; parts of the source code are also presented using the C++ programming language.

Keywords: RBF Networks, Optimization, Genetic Algorithm

Περιεχόμενα

Κατάλογος εικόνων

Κατάλογος αλγορίθμων

Κατάλογος πινάκων

1. Εισαγωγή

Τα radial basis function δίκτυα (RBF) είναι τεχνητά νευρωνικά δίκτυα πρόσθιας τροφοδότησης (feedforward) που χρησιμοποιούνται για προβλήματα προσέγγισης συναρτήσεων, classification ή και regression.

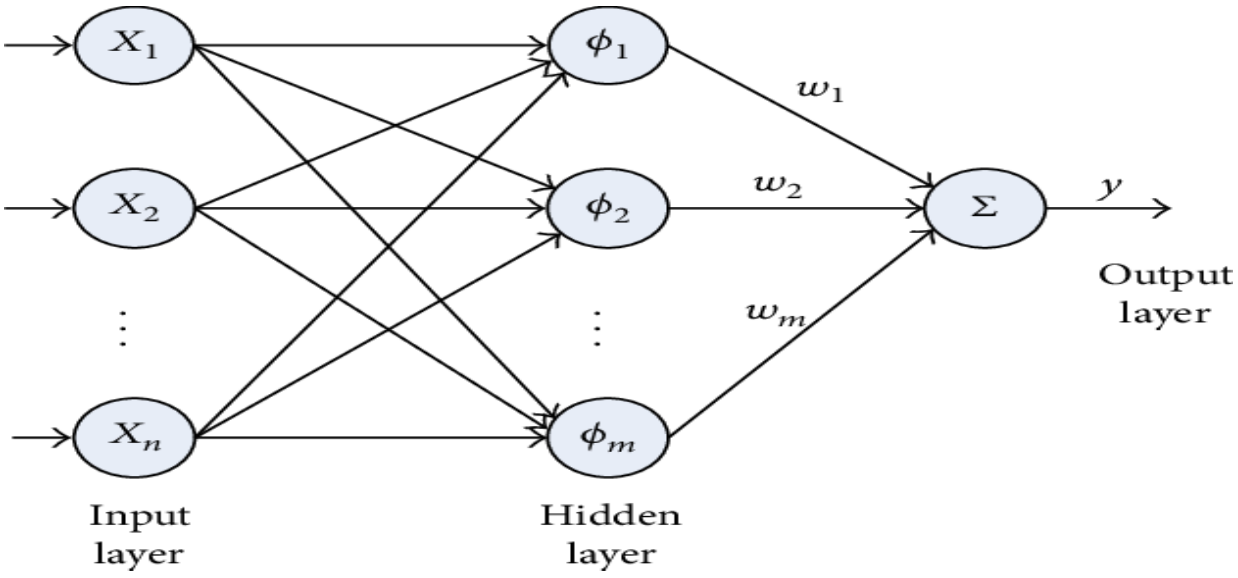
Η βασική διάταξη ενός τέτοιου δικτύου αποτελείται από τρία στρώματα (layers) .

- ένα στρώμα εισόδου (input layer).
- ένα στρώμα επεξεργασίας ή κρυφό στρώμα (hidden layer).
- ένα στρώμα εξόδου (output layer).

Η κυρίως επεξεργασία γίνεται στο μεσαίο στρώμα στο οποίο βασική προϋπόθεση είναι να ορίσουμε μια ακτινική συνάρτηση ως συνάρτηση ενεργοποίησης, η οποία συνήθως επιλέγεται να είναι η συνάρτηση Gauss. Στην συγκεκριμένη εργασία το δίκτυο υλοποιείται σε γλώσσα προγραμματισμού C++ και εκπαιδεύεται σε δύο φάσεις.

Στην πρώτη, τα κέντρα c και η διακύμανση σ^2 προσδιορίζονται χρησιμοποιώντας μια τεχνική clustering που στην περίπτωση αυτή είναι ο αλγόριθμος K-Means ο οποίος θα αναλυθεί σε παρακάτω ενότητα. Στην δεύτερη φάση, τα βάρη μεταξύ του output και hidden layer υπολογίζονται με γνώμονα την ελαχιστοποίηση μιας τιμής κόστους όπως το Mean Squared Error για το συγκεκριμένο Dataset. Για να εκπαιδευθεί το δίκτυο και να υπολογιστούν τα βάρη, στο εξωτερικό στρώμα επιβάλλεται να λυθεί ένα σύστημα εξισώσεων το οποίο αναλύεται σε επόμενη ενότητα.

Επίσης, παρουσιάζεται το συγκεκριμένο θέμα ως ένα optimization πρόβλημα του οποίου η βέλτιστη λύση μπορεί να βρεθεί με την χρήση γενετικού αλγορίθμου. Αντιμετωπίζουν τα δεδομένα σαν έναν πληθυσμό από χρωμοσώματα όπου κάθε ένα ισοδυναμεί με μία πιθανή λύση στο πρόβλημα. Σε κάθε δομή δεδομένων που αποτελεί ένα χρωμόσωμα ανατίθεται μια τιμή καταλληλόλητας (fitness score) που με βάση αυτή προσδιορίζεται η καταλληλόλητα της συγκεκριμένης λύσης. Στα χρωμοσώματα με μεγαλύτερο fitness score δίνεται η ευκαιρία για αναπαραγωγή προσπαθώντας έτσι να βρεθεί η βέλτιστη λύση. [1],[2]



Εικόνα 1. Η βασική διάταξη ενός RBF δικτύου

2. Radial Basis Function Networks

Απώτερος σκοπός ενός νευρωνικού δικτύου είναι η “μάθηση” που ουσιαστικά αποτελεί πρόβλημα προσέγγισης, το οποίο σχετίζεται με γενικότερα προβλήματα εκτίμησης τιμών όπως η παρεμβολή (interpolation) σε ένα σύνολο δεδομένων. Συνολικά το RBF αποτελεί δημοφιλής λύση για τέτοιου είδους προβλήματα μιας και έχει απλούστερη δομή και μπορεί να εκπαιδεύεται πολύ πιο γρήγορα. Στο συγκεκριμένο κεφάλαιο θα αναλυθεί η δομή του αλλά και το training process που ακολουθεί.

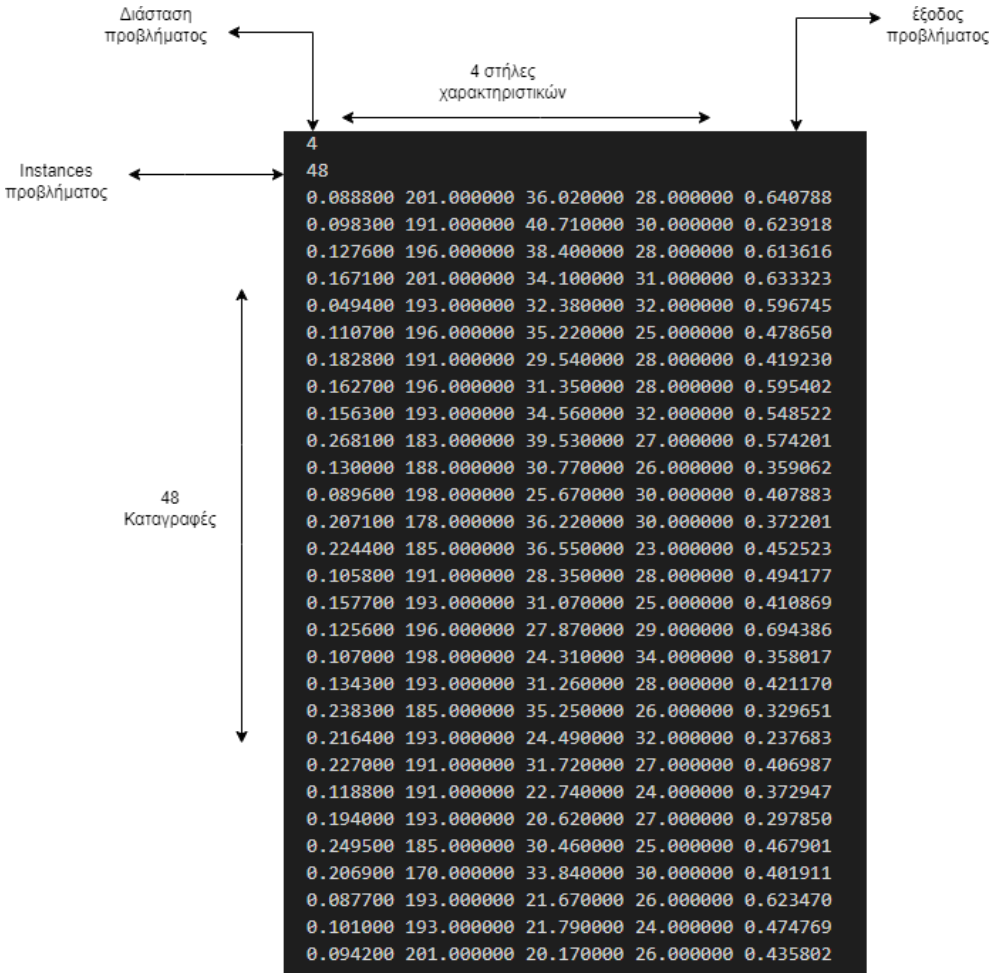
2.1 Input Layer

Το στρώμα εισόδου τροφοδοτεί δεδομένα από το dataset, δεν εκτελείται κάποιος υπολογισμός και απλά συνδέονται με τους κρυμμένους νευρώνες. Κάθε dataset αποτελείτε από χαρακτηριστικά που στο σύνολό τους αποτελούν πρότυπα. Τα χαρακτηριστικά μπορούν να διακρίνονται σε :

- Συνεχείς τιμές
- Διακριτές τιμές
- Αλφαριθμητικές τιμές

Σε ένα dataset μπορεί να υπάρχουν και δύο διαφορετικές κατηγορίες χαρακτηριστικών, με το κάθε σύνολο δηλαδή πρότυπο να είναι μια ξεχωριστή καταγραφή. Απαραίτητη προϋπόθεση είναι να γνωρίζουμε τον αριθμό των προτύπων αλλά και την διάσταση καθενός από αυτά διότι βάση αυτών ρυθμίζονται παράμετροι για το RBF αλλά και τον γενετικό αλγόριθμο. Τέλος, αξίζει να αναφερθεί πως ο μετασχηματισμός των δεδομένων από το στρώμα εισόδου στο στρώμα επεξεργασίας είναι μη γραμμικός (non-linear).

Η μορφή του dataset φαίνεται στην εικόνα 2. Αρχικά το αρχείο αποτελείτε από 2 ακέραιους αριθμούς οι οποίοι είναι η διάσταση του συγκεκριμένου προβλήματος ενώ ο δεύτερος αποτελεί τον αριθμό από στοιχεία. Κάθε γραμμή αποτελεί ένα πρότυπο, δηλαδή μια ξεχωριστή καταγραφή. Στην συγκεκριμένη περίπτωση εφόσον η διάσταση του προβλήματος είναι ίση με 4, θα υπάρχουν 4 στήλες με χαρακτηριστικά και η $5^{η}$ στήλη θα αποτελεί τον χαρακτηρισμό για το συγκεκριμένο πρόβλημα ή αλλιώς την έξοδο του προτύπου.



Εικόνα 2. Διάταξη dataset

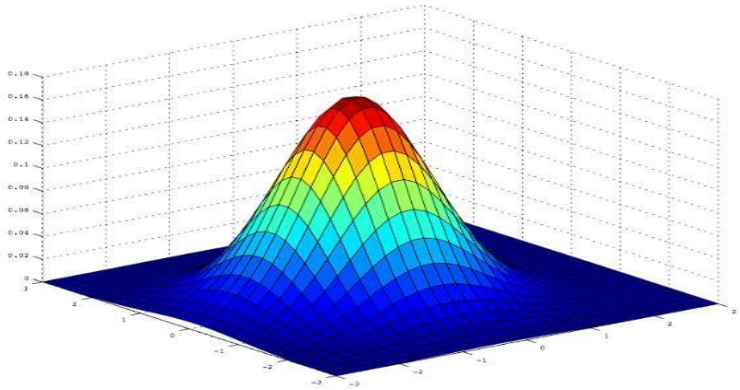
2.2 Hidden Layer

Η βασική διάταξη ενός RBF δικτύου όπως προαναφέραμε αποτελείται από τρία στρώματα. Το στρώμα εισόδου απλά μεταφέρει και τροφοδοτεί το στρώμα επεξεργασίας το οποίο περιέχει έναν αριθμό από μη γραμμικούς νευρώνες επεξεργασίας οι οποίοι χρησιμοποιούν και έχουν ως έξοδο την ίδια ακτινική συνάρτηση.

Μια συνάρτηση λέγεται συνάρτηση ακτινικού τύπου (radial function) αν υπάρχει ένα διάνυσμα c που ονομάζεται κέντρο (center ή centroid) και η τιμή της συνάρτησης εξαρτάται από την απόσταση του x από το κέντρο c. Δηλαδή : f(x)= f(d(x-c)).

Συνήθως στις υλοποιήσεις των RBF χρησιμοποιείται η συνάρτηση Gauss :

$$f(x) = e^{-\frac{d^1(x-c)^2}{2\sigma^2}}$$



Εικόνα 3. Plot της συνάρτησης Gauss

Για να υπολογισθεί η έξοδος του hidden layer εκτός της εισόδου χρειάζεται και η εύρεση των c και σ^2 όπου μπορούν να βρεθούν με διαφορετικούς τρόπους. Για παράδειγμα μια προσέγγιση που θεωρείται όμως μη βέλτιστη είναι να προσδιορίσουμε τα κέντρα c τυχαία ενώ μια πιο σωστή λύση είναι να γίνει χρήση μιας clustering τεχνικής όπως ο αλγόριθμος K-Means. Σε αντίθεση με την μη γραμμικότητα των δύο πρώτων στρωμάτων τα δεδομένα από το στρώμα επεξεργασίας προς το εξωτερικό στρώμα μετασχηματίζονται γραμμικά (linear). [1],[2]

[1] $d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2}$

2.2.1 K-Means Αλγόριθμος

Ο αλγόριθμος Κ μέσων χρησιμοποιείται στο πρώτο στάδιο του RBF δικτύου έτσι ώστε να υπολογιστούν τα κέντρα με ακρίβεια. Αποτελεί αλγόριθμο ο οποίος πετυχαίνει Partitioning Clustering. Σχετίζει το κάθε cluster με ένα κεντρικό σημείο (centroid) και το κάθε σημείο ανατίθεται στο cluster με την μικρότερη απόσταση προς το centroid. Βασική προϋπόθεση για τον αλγόριθμο K-Means είναι ότι ο αριθμός από clusters Κ πρέπει να έχει ήδη οριστεί. Για την μέτρηση της απόστασης αλλά και για το που θα γίνει assigned ένα σημείο χρησιμοποιείται η ευκλείδεια απόσταση ενώ για την αξιολόγηση της τοποθέτησης σημείων σε ένα cluster χρησιμοποιείται μια συνάρτηση κόστους βάση της απόστασης σημείων προς το cluster. Έτσι εάν έχουμε ένα σημείο και δύο clusters θα γίνει επιλογή αυτού με το λιγότερο error.

Για την επιλογή του Κ μπορούμε να τρέξουμε τον αλγόριθμο με διαφορετικό αριθμό από Κ και να εξάγουμε τον καλύτερο ή ακόμα και να χρησιμοποιήσουμε διαφορετική τεχνική για τον υπολογισμό του όπως τον Expectation-Maximization αλγόριθμο. [3]

K-Means Algorithm

1. Αρχικοποίηση N κέντρων : c_i με $i = 1 \ldots N$
2. Επανέλαβε :
   - : • $S_i = \{\ \}, i = 1 \ldots N$
   - : • Εύρεση ομάδας για κάθε πρότυπο
   - : • Έστω M_j το πλήθος των μελών ομάδας, x_i τα μέλη ομάδας ανανέωσε τα κέντρα ως εξής:
     - $c_j = \frac{1}{M_j} \sum_{i=1}^{Mj} x_i$
3. Αν τα κέντρα δεν έχουν αλλάξει τότε **τερματισμός** αλλιώς μετάβαση στο βήμα 2. :
4. Υπολογισμός διακύμανσης σ ως: $\sigma = \frac{d}{\sqrt{2s}}$

Αλγόριθμος 1. K-Means

Μερικά προβλήματα του αλγορίθμου :

- Δυσκολεύεται να δώσει καλά αποτελέσματα όταν τα clusters που πρέπει να σχηματίσει έχουν μη σφαιρικό σχήμα αλλά και μεγάλη διαφορά σε μέγεθος και πυκνότητα.
- Δεν δίνει σωστές λύσεις όταν το αρχικό Dataset περιέχει ακραίες τιμές.

2.3 Output Layer

Ο υπολογισμός στο εξωτερικό επίπεδο είναι ο κλασσικός που συναντάμε σε νευρωνικά δίκτυα που στην ουσία αποτελεί έναν γραμμικό συνδυασμό μεταξύ του διανύσματος εισόδου και του διανύσματος βάρους.

Εάν έχει επιλεγεί η συνάρτηση Gauss ως συνάρτηση ενεργοποίησης για τους νευρώνες τότε μπορούμε να προσδιορίσουμε την έξοδο του δικτύου ως εξής :

$$f(x) = \sum_{i=1}^{m} W_j H_j(x)$$

Με W_j το διάνυσμα βαρών στο οποίο πρέπει να γίνει μάθηση με λύση ενός γραμμικού συστήματος.

Θεωρούμε τον πίνακα A με H(x) την έξοδο του hidden layer :

$$A = \begin{bmatrix} H(x_1, c_1) & H(x_1, c_2) & H(x_1, c_3) \\ H(x_2, c_1) & H(x_2, c_2) & H(x_2, c_3) \\ \dots & \dots & \dots \\ H(x_i, c_1) & H(x_i, c_2) & H(x_i, c_3) \end{bmatrix}$$

Τα βάρη ενημερώνονται με βάση την πράξη : $w = (A^T A)^{-1} * A^T y$. Ουσιαστικά πολλαπλασιάζεται ο πίνακας $(A^T A)^{-1} * A^T$ ο οποίος ονομάζεται ψευδοαντίστροφος με την έξοδο του δικτύου RBF δηλαδή το y. [1],[2]

Για να βρεθεί ο ψευδοαντίστροφος χρειαζόμαστε τρείς πράξεις μεταξύ πινάκων :

- Πολλαπλασιασμός πινάκων
- Υπολογισμός ανάστροφου πίνακα (Transpose)
- Υπολογισμός αντίστροφου πίνακα (Inverse)

Οι παραπάνω πράξεις υλοποιούνται ως συναρτήσεις :

- Matrix rbf::Multiply(Matrix &A, Matrix &B)
- Matrix rbf::Transpose(Matrix &A)
- Matrix rbf::Inverse(Matrix &mat)

Έπειτα εφόσον έχουμε υλοποιήσει τις παραπάνω μεθόδους απλά λύνουμε το γραμμικό σύστημα.

Συνολικά το γεγονός ότι υπάρχει ένα μόνο κρυφό επίπεδο στο μοντέλο αρχιτεκτονικής, κάνει την υλοποίηση της συγκεκριμένης δομής δικτύου αρκετά απλούστερη. Επίσης, δεν χρειάζονται περεταίρω παράμετροι από αυτούς που προαναφέραμε για την εκπαίδευσή του γεγονός που μειώνει τον απαιτούμενο όγκο υπολογισμών αλλά εγγυάται και την σταθερότητα του δικτύου.

2.4 Συνάρτηση κόστους

Ως συνάρτηση κόστους για το δίκτυο χρησιμοποιούμε το άθροισμα των τετραγώνων των σφαλμάτων (Sum Squared Error) το οποίο υπολογίζεται σύμφωνα με την εξίσωση :

$$SSE = \sum_{i=1}^{p} (y - f(x_i))^2$$

Το SSE ουσιαστικά υπολογίζει την διαφορά μεταξύ της εξόδου που παράγει το δίκτυο και της εξόδου που υπάρχει στο εκάστοτε Dataset. Με άλλα λόγια, μετρά την διακύμανση σφάλματος και υποδουλώνει κατά πόσο ένα regression μοντέλο είναι ικανό να δώσει καλά αποτελέσματα για ένα σύνολο δεδομένων. Εάν το SSE είναι μεγάλος αριθμός σημαίνει πως το μοντέλο δεν μπορεί να εξηγήσει τα δεδομένα και δεν δίνει ιδεατή λύση. Στην συγκεκριμένη υλοποίηση κάνουμε output το SSE_{train} και το SSE_{test} για το train και test dataset αντίστοιχα.[4]

Επίσης, υπολογίζεται και το Classification error σύμφωνα με τον αλγόριθμο :

Classification Error Algorithm

1. Έστω N πρότυπα με κατηγορία/έξοδο Y , μετρητής missed=0 :
2. Επανέλαβε :
   - Για κάθε ξεχωριστή κατηγορία Y_j εύρεση MIN,MAX του Gauss(X_j)
   - Για κάθε πρότυπο X_i υπολόγισε το Gauss(X_i)
   - Εάν για i πρότυπο ισχύει : $\min j \leq Gauss(X_i) \geq \max j$ τότε τοποθέτησε το πρότυπο X_i στην συγκεκριμένη j κατηγορία.
   - Εάν $j \neq Y_i$ κάνε missed=missed+1
3. Τέλος επανάληψης :
4. Υπολόγισε το Classification Error ως $\frac{missed}{N}$:

Αλγόριθμος 2. Classification error

Και τα τρία παραπάνω error υπολογίζονται μέσω τριών συναρτήσεων :

- SSE_{train}: double rbf::TrainSumSquaredError()
- SSE_{test}: double rbf::TestSumSquaredError()
- Classification error: void rbf::MinimumClassificationError()

2.5 Πειραματικά αποτελέσματα

Θα χρησιμοποιήσουμε γνωστά classification και regression datasets[2] παρουσιάζοντας την σημασία της επιλογής βέλτιστου αριθμού από νευρώνες για το μεσαίο επίπεδο.

Συγκεκριμένα :

- Wine Dataset, το οποίο παρέχει αποτελέσματα χημικής ανάλυσης κρασιών.
- WDBC Dataset, που παρέχει δεδομένα από όγκους του μαστού.
- Thyroid Dataset, το οποίο περιέχει δεδομένα παθήσεων του θυρεοειδούς.
- Ionosphere Dataset, το οποίο περιέχει δεδομένα ραντάρ από σύστημα κεραιών στο Goose Bay.
- Iris Dataset, περιέχει δεδομένα που αφορούν το φυτό Ίριδας. [5]

Dataset	**N=2**	**N=4**	**N=6**	**N=8**	**N=10**
Wine	5.03423	4.4601	4.5143	4.7055	4.5863
WDBC	9.91926	6.8503	6.6388	5.7275	5.5989
Thyroid	453.0078	389.5216	322.6643	292.9250	299.1580
Ionosphere	37.7404	36.0976	35.5272	35.8274	36.2297
Iris	24.1189	13.1986	7.6664	7.1876	6.1841

Πίνακας 1. Sum Squared Train Error

Από τους πίνακες προκύπτει ότι από N=6 και μετά το error μειώνεται αλλά με λιγότερα rate, ενώ σε ορισμένες περιπτώσεις ένας μεγαλύτερος αριθμός από N νευρώνες θα δώσει εάν όχι ίδια χειρότερα αποτελέσματα. Επομένως, κρίνεται βέλτιστη λύση το N να κυμανθεί από 6-8 έτσι ώστε να μην δημιουργούνται αριθμητικά προβλήματα στον υπολογισμό του ψευδοαντίστροφου (σε μεγάλο αριθμό από N) αλλά και το δίκτυο να δίνει λύση σε ικανοποιητική ταχύτητα.

[2] Τα παραπάνω datasets είναι διαθέσιμα στο repository από το UCI. [5]

2.6 Διαφορετικές τεχνικές εκπαίδευσης ενός RBF δικτύου

- One-Phase Learning

Στην συγκεκριμένη τεχνική η μοναδική παράμετρος του δικτύου η οποία ρυθμίζεται με γνώμονα την ελαχιστοποίηση του σφάλματος είναι το διάνυσμα των βαρών. Οι υπόλοιπες παράμετροι όπως τα κέντρα ρυθμίζονται μέσω του Dataset εισόδου χωρίς κάποιον αλγόριθμο εκπαίδευσης (πχ K-Means), για παράδειγμα μια προσέγγιση είναι ότι εάν έχουμε επιλέξει N αριθμό από νευρώνες και χρειαζόμαστε C_N κέντρα, αυτά προσεγγίζονται με τυχαίες τιμές από το dataset εισόδου. Κάτι ανάλογο συμβαίνει και για υπόλοιπες παραμέτρους του δικτύου όπως η διακύμανση η οποία αρχικοποιείται σε μια σταθερή τιμή εξ αρχής.

- Two-Phase Learning

Αποτελεί την τεχνική στην οποία βασίζεται η συγκεκριμένη εργασία. Το στρώμα επεξεργασίας και το στρώμα εξόδου εκπαιδεύονται ξεχωριστά σε διαφορετικές φάσεις, πρώτα ορίζονται τα κέντρα c_{ij} και η διακύμανση σ και έπειτα ρυθμίζεται η έξοδος. Σε αντίθεση με το one-phase learning αντί για τυχαία τοποθέτηση των κέντρων, η συγκεκριμένη τεχνική στοχεύει σε μεγαλύτερη ακρίβεια ως προς την αρχικοποίηση των κέντρων. Αυτό επιτυγχάνεται με κάποια τεχνική clustering όπως ο αλγόριθμος k-means ή μια άλλη τεχνική είναι να προσεγγίσουμε τα κέντρα μέσω των δέντρων αποφάσεων. Τέλος, εφόσον τα κέντρα έχουν υπολογισθεί η διακύμανση σ προσεγγίζεται ως $\sigma = \sqrt{\frac{d}{2s}}$ όπου d η απόσταση μεταξύ των πιο απομακρυσμένων κέντρων και s ο αριθμός από κέντρα.

- Three-Phase Learning

Η τεχνική αυτή ακολουθεί την εκπαίδευση δύο φάσεων για να ορίσει τις βασικές παραμέτρους του δικτύου και έπειτα όλη η δομή περνάει μέσα από μια μέθοδο περαιτέρω βελτιστοποίησης. Όπως προαναφέραμε, η διφασική εκπαίδευση πετυχαίνει μεγαλύτερη ακρίβεια ως προς την τυχαία προσέγγιση παραμέτρων της πρώτης περίπτωσης. Αυτό σημαίνει πως είναι μια αρκετά γρήγορη μέθοδος εκπαίδευσης του δικτύου η οποία ωστόσο σε ορισμένες περιπτώσεις καταλήγει σε μη ιδανικά αποτελέσματα ταξινόμησης. Γι' αυτό τον λόγο η τριφασική εκπαίδευση υιοθετεί τρόπους παραπλήσιους του MLP όπως το back propagation για περαιτέρω μείωση συνάρτησης κόστους άρα και επίτευξης μεγαλύτερης ακρίβειας. [6]

2.7 Feed Forward νευρωνικό δίκτυο

Ένα νευρωτικό δίκτυο ονομάζεται πρόσθιας τροφοδότησης (Feed Forward) εάν οι κόμβοι του δεν σχηματίζουν κάποιο κύκλο. Είναι η απλούστερη μορφή μεταφοράς πληροφορίας μεταξύ των δικτύων μιας και αυτή ενώ μπορεί να περνάει από διάφορα layers ανάλογα τον τύπο δικτύου που χρησιμοποιείται κινείται μόνο προς την κατεύθυνση του output. Συχνά σχηματίζονται από 3 επίπεδα (input, hidden και output) και τα δεδομένα εισέρχονται στους κόμβους εισόδου, περνούν προς το hidden layer και τελικά εξέρχονται από τους κόμβους εξόδου.

Το δίκτυο δεν διαθέτει συνδέσεις οι οποίες θα επέτρεπαν στις πληροφορίες που εξέρχονται από το εξωτερικό στρώμα να σταλούν προς τα προηγούμενα επίπεδα. Τέτοιου είδους δίκτυα χρησιμοποιούνται για αναγνώριση και ταξινόμηση προτύπων, την μη γραμμική παλινδρόμηση αλλά και την προσέγγιση συναρτήσεων. [7]

2.8 Σύγκριση RBF με MLP

Ένας άλλος τύπος νευρωνικού δικτύου ο οποίος μοιάζει με τα RBF μιας και είναι feedforward αλλά και μη γραμμικός είναι το MLP (Multilayer Perceptron).Επομένως το ένα μπορεί να μιμηθεί το άλλο με αρκετά μεγάλη ακρίβεια πράγμα όμως που δεν αναιρεί το γεγονός ότι διαφέρουν σε πολλά σημεία:

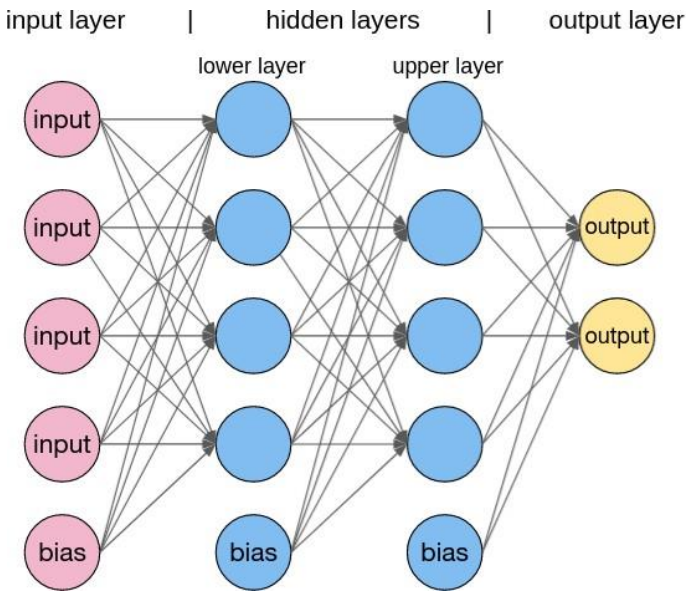
- Ένα MLP δίκτυο μπορεί να έχει περισσότερα από ένα hidden layers σε αντίθεση με το RBF το οποίο στην συνηθέστερη μορφή έχει μόνο ένα.

- Ένα MLP δίκτυο είναι πιθανό να έχει στρώμα επεξεργασίας στο hidden ή output layer σε αντίθεση με ένα δίκτυο RBF στο οποίο το hidden και output layer εξυπηρετούν τελείως διαφορετικούς σκοπούς.

- Το output ή hidden layer σε ένα MLP μπορεί να είναι linear ή non-linear ανάλογα το πρόβλημα σε αντίθεση με το RBF του οποίου το output layer είναι πάντα linear.

- Χρησιμοποιούν διαφορετικές συναρτήσεις ενεργοποίησης.

- Ένα RBF δίκτυο είναι δυνατόν να υλοποιεί non linear μετασχηματισμούς στα δεδομένα εισόδου. [1],[8]



Εικόνα 4. Διάταξη MLP δικτύου

2.9 Τρόποι βελτίωσης απόδοσης του RBF

Εκτός από το να κάνουμε optimize την έξοδο του RBF, πράγμα που θα αναλύεται στο 3.2 μπορούμε να βελτιώσουμε την βασική οργάνωση του δικτύου με σκοπό να έχουμε καλύτερα αποτελέσματα.

- Ο αριθμός από νευρώνες *κ* στο κρυμμένο επίπεδο δεν θα πρέπει να είναι ίσος με την διάσταση N του εκάστοτε Dataset (σε ορισμένες περιπτώσεις επιδιώκεται το M να είναι αρκετά χαμηλότερος αριθμός από το N).

- Τα κέντρα δεν πρέπει να οριστούν ως τα δεδομένα εισόδου ή τυχαία αλλά χρησιμοποιώντας κάποιον αλγόριθμο εκπαίδευσης.

- Είναι πιο αποτελεσματικό να έχουμε διαφορετικές διακυμάνσεις σ οι οποίες μπορούν να βρεθούν με κάποιον αλγόριθμο εκπαίδευσης.

- Μπορούμε να εισάγουμε παράμετρο bias πράγμα που θα αντισταθμίσει την διαφορά μεταξύ των εξόδων του δικτύου έναντι των target εξόδων.

Συνολικά τα δίκτυα RBF έχουν ευκολότερη σχεδίαση και μεγάλη ανοχή στον θόρυβο πράγμα που τα καθιστά κατάλληλα για τον σχεδιασμό ευέλικτων συστημάτων. [9]

3. Εισαγωγή στους γενετικούς αλγορίθμους

Οι γενετικοί Αλγόριθμοι είναι μέθοδοι που μπορούν εύκολα να προσαρμόζονται έτσι ώστε να χρησιμοποιηθούν για την επίλυση προβλημάτων βελτιστοποίησης. Βασίζονται στην θεωρία εξέλιξης των ειδών και στις ενέργειες βιολογικών οργανισμών. Οι γενετικοί αλγόριθμοι στηρίζονται στις αρχές της επιβίωσης του ικανότερου και της φυσικής επιλογής. Χρησιμοποιώντας αυτό το μοντέλο είναι ικανοί να βελτιστοποιήσουν λύσεις σε πραγματικά προβλήματα στον τομέα της μηχανικής μάθησης. Προτάθηκαν αρχικά από τον Holland και η βασική ιδέα είναι ότι στην φύση άτομα σε έναν πληθυσμό ανταγωνίζονται για πόρους με αποτέλεσμα τα πιο επιτυχημένα άτομα στην επιβίωση με βάση μια τιμή καταλληλόλητας να έχουν περισσότερες πιθανότητες να αναπαράγουν απογόνους (offsprings). [10],[11],[12]

Πλεονεκτήματα χρήσης γενετικών αλγορίθμων :

- Παρέχει την δυνατότητα παραλληλισμού δεδομένων
- Καθολική βελτιστοποίηση
- Παρέχει μεγαλύτερο σύνολο τιμών για λύση
- Απαιτεί λιγότερες πληροφορίες
- Παρέχει πολλαπλές βέλτιστές λύσεις
- Αναπαριστά τα δεδομένα με την χρήση γενετικών χρωμοσωμάτων

Μειονεκτήματα χρήσης γενετικών αλγορίθμων :

- Είναι κατά κύριο λόγο πολύ πιο αργός στην εκτέλεση από άλλες εναλλακτικές μεθόδους όπως οι μέθοδοι Reduced Gradient και Simplex
- Αδυνατεί να βρει κάτι κοντά σε μία βέλτιστη λύση όταν το πρόβλημα είναι μεγάλο και εισάγονται πολλές μεταβλητές
- Δεν υπάρχει τρόπος να ελέγχει εάν μια λύση είναι τοπικά βέλτιστη μιας και εάν ανακαλύψει μια «καλή» είναι γιατί την σύγκρινε με προηγούμενες υποψήφιες [13]

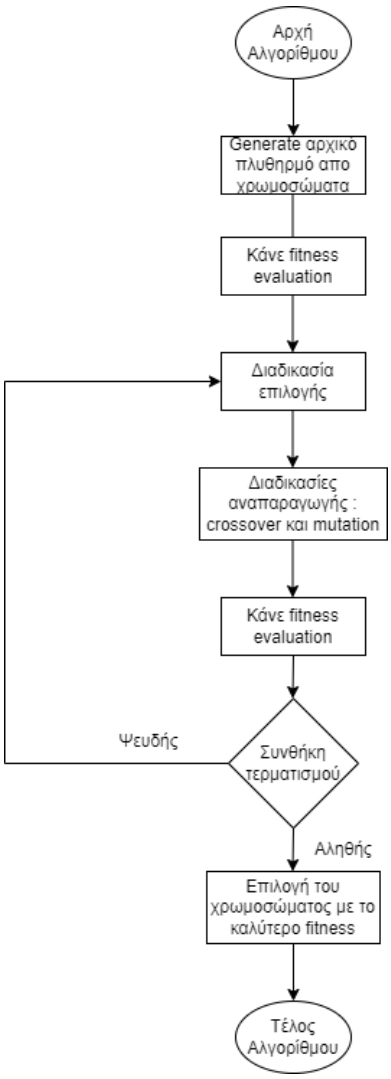
3.1 Εισαγωγικές έννοιες

Ο αλγόριθμος παρακάτω παρουσιάζει τις βασικές ενέργειες που εκτελούνται σε έναν γενετικό αλγόριθμό.

Τα βασικά στοιχεία απαραίτητα για την λειτουργία ενός γενετικού αλγορίθμου είναι τα εξής :

- Έναν αρχικό πληθυσμό από χρωμοσώματα.
- Fitness Function (Μια συνάρτηση καταλληλόλητας για να μπορεί να γίνει κάποια αξιολόγηση και να φτάσουμε σε κάποια optimized λύση).
- Μια διαδικασία επιλογής των πιο ικανών χρωμοσωμάτων.
- Διαδικασίες αναπαραγωγής : crossover και mutation.

Κάθε βήμα αλλά και στοιχείο αναλύεται στα επόμενα υποκεφάλαια όπου δίνεται μια γενικότερη ιδέα για το πως λειτουργεί ένας τέτοιος αλγόριθμος. Στο κεφάλαιο [3.2] προσαρμόζουμε έναν γενετικό αλγόριθμο για να κάνουμε optimize την λύση του RBF δικτύου. [13]



Εικόνα 5. Βασικές ενέργειες Γενετικών Αλγορίθμων

3.1.1 Αρχικός πληθυσμός

Στο πρώτο βήμα, αρχικοποιούνται τα άτομα (individuals) σχηματίζοντας ένα σύνολο το οποίο ονομάζεται αρχικός πληθυσμός, με το κάθε άτομο να αποτελεί μια λύση στο εκάστοτε πρόβλημα. Η κάθε λύση αναλύεται σε ένα σύνολο από παραμέτρους που ονομάζονται γονίδια (genes). Τα γονίδια ενώνονται μαζί για να σχηματίσουν μια ακολουθία από τιμές που ονομάζεται χρωμόσωμα.

Μπορούμε να κατηγοριοποιήσουμε τα encoding schemes ανάλογα με τον τρόπο λειτουργίας τους, δηλαδή :

- Encoding schemes τα οποία η καταλληλόλητα χρωμοσωμάτων εξαρτάται μόνο από την σειρά των γονιδίων, πχ Permutation encoding
- Encoding schemes τα οποία η καταλληλόλητα χρωμοσωμάτων εξαρτάται και από την σειρά αλλά και την τιμή των γονιδίων, πχ Binary encoding
- Encoding schemes τα οποία η καταλληλόλητα χρωμοσωμάτων εξαρτάται μόνο από την τιμή των γονιδίων, πχ Value encoding

Η ένωση αυτή των γονιδίων (encoding) γίνεται με διαφορετικούς τρόπους :

- Binary Encoding

Είναι η πιο γνωστή και σύνηθες τεχνική encoding κατά την οποία το χρωμόσωμα κωδικοποιείται σε δυαδικό σύστημα.

1	0	1	0	1	1

- Permutation Encoding

Χρησιμοποιείται συνήθως σε προβλήματα με γραφήματα, με το κάθε χρωμόσωμα να είναι μια ακολουθία από αριθμούς που αναπαριστούν κόμβους.

5	9	1	4	2	2

- Value Encoding

Τεχνική που αναπαριστά το χρωμόσωμα μια συμβολοσειρά από τιμές. Η τιμή μπορεί να είναι ακέραιος, πραγματικός αριθμός ή χαρακτήρας και χρησιμοποιείται σε προβλήματα όπου η δυαδική αναπαράσταση δεν δίνει λύση.

5.123	9.543	1.654	4.876	2.234	2.145

[14]

3.1.2 Συνάρτηση καταλληλόλητας

Σε αυτό το βήμα πρέπει να εφαρμοστεί μια συνάρτηση καταλληλόλητας για το εκάστοτε πρόβλημα με σκοπό να αξιολογήσει (evaluate) το κάθε χρωμόσωμα. Η συνάρτηση αυτή επιστρέφει μια τιμή που είναι ανάλογη με την χρησιμότητα και καταλληλόλητα του συγκεκριμένου χρωμοσώματος. Έτσι υπολογίζεται το fitness για κάθε χρωμόσωμα και μετά γίνεται το evaluation με σκοπό να ξεχωρίσουμε τα πιο ικανά τα οποία θα έχουν και μεγαλύτερη πιθανότητα αναπαραγωγής.

Απαραίτητες προϋποθέσεις για την συνάρτηση καταλληλόλητας :

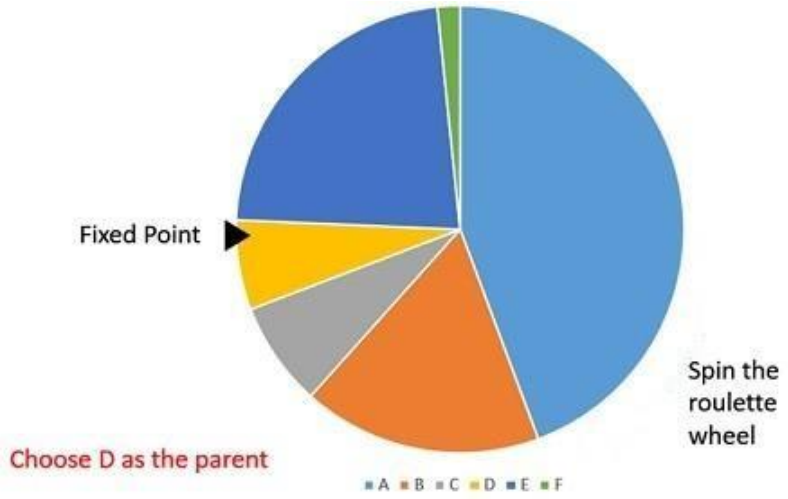
- Η fitness function θα πρέπει να έχει υλοποιηθεί αποτελεσματικά ειδάλλως εφόσον για κάθε επανάληψη γίνεται evaluation και η μέθοδος αυτή είναι αργή στην εκτέλεση, αυτό θα επηρεάσει πολύ αρνητικά την ταχύτητα εκτέλεσης του γενικότερου αλγόριθμου.
- Θα πρέπει να επιστρέφει μια τιμή η οποία θα καθορίζει με σαφήνεια πόσο κατάλληλη είναι μία λύση για την επίλυση του εκάστοτε προβλήματος
- Εάν ο υπολογισμός είναι δύσκολος λόγο της πολυπλοκότητας του προβλήματος θα πρέπει να γίνει μια προσέγγιση της τιμής με σκοπό οι καλύτερες/χειρότερες λύσεις να έχουν και τις καλύτερες/χειρότερες τιμές καταλληλόλητας. [15]

3.1.3 Διαδικασία επιλογής

Το Selection phase αποτελεί την διαδικασία επιλογής των καταλληλότερων χρωμοσωμάτων από τον πληθυσμό τα οποία θα χρησιμοποιηθούν για αναπαραγωγή. Κάποια παραδείγματα μεθόδων για επιλογή :

- Roulette Wheel Selection

Μπορούμε να φανταστούμε τα χρωμοσώματα με την ανάλογη fitness τιμή σε ένα pie chart στο οποίο ορίζουμε ένα σταθερό σημείο. Όποια περιφέρεια χρωμοσώματος τύχει στο fixed σημείο επιλέγεται σαν γονέας και χρησιμοποιείται για αναπαραγωγή.

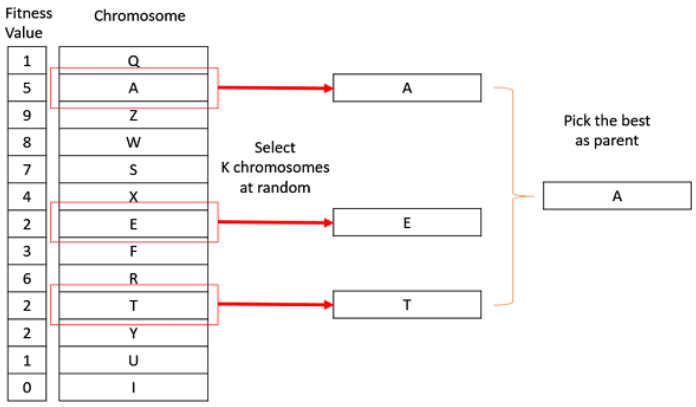


Chromosome	Fitness Value
A	8.2
B	3.2
C	1.4
D	1.2
E	4.2
F	0.3

Εικόνα 6. Roulette Wheel Selection

- Tournament Selection

Σε αυτή την προσέγγιση, επιλέγουμε τυχαία Ν αριθμό χρωμοσωμάτων από τα οποία εξάγουμε το καλύτερο βάση του fitness.



Εικόνα 7. Tournament Selection

- Random Selection

Είναι πιο απλή στην υλοποίηση μιας και δεν επιλέγει χρωμοσώματα για αναπαραγωγή με βάση του fitness αλλά τυχαία πράγμα που δεν δίνει τόσο καλά αποτελέσματα οπότε συνήθως αποφεύγεται. [16]

3.1.4 Διαδικασία αναπαραγωγής

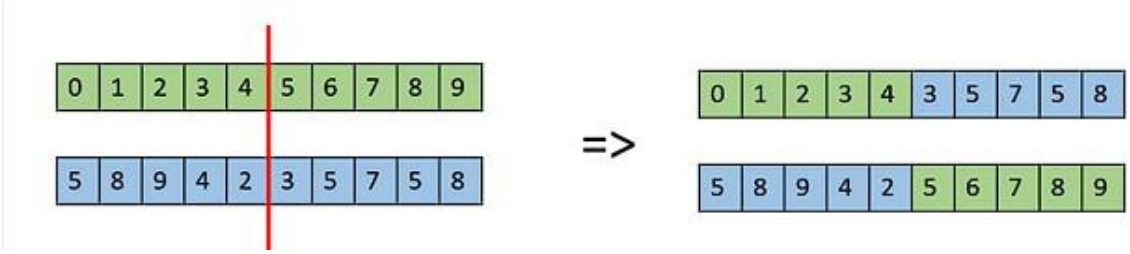
Στην φάση της αναπαραγωγής αφού έχουν επιλεγεί τα κατάλληλα χρωμοσώματα από την προηγούμενη φάση, συνδυάζονται με σκοπό τα νέα παραγόμενα να δίνουν καλύτερο fitness. Ο συνδυασμός αυτός γίνεται με έναν μηχανισμό διασταύρωσης (crossover) και μετάλλαξης (mutation). Θα αναλύσουμε κάποιους βασικούς τρόπους crossover και mutation.

Crossover Τεχνικές :

Ο μηχανισμός αυτός είναι ένα πολύ ισχυρό εργαλείο για την εισαγωγή νέων στοιχείων στον πληθυσμό διατηρώντας παράλληλα ποικιλότητα, σε συνδυασμό με το γεγονός πως θα έχουν επιλεγεί τα καλύτερα χρωμοσώματα είναι αποδεδειγμένο πως θα παραχθούν καλοί ή ακόμα και καλύτεροι απόγονοι. Η ανταλλαγή αυτή γονιδίων μεταξύ των χρωμοσωμάτων μπορεί να γίνει με ποικίλους τρόπους, παρακάτω παρουσιάζουμε κάποιους από τους πιο σύνηθες.

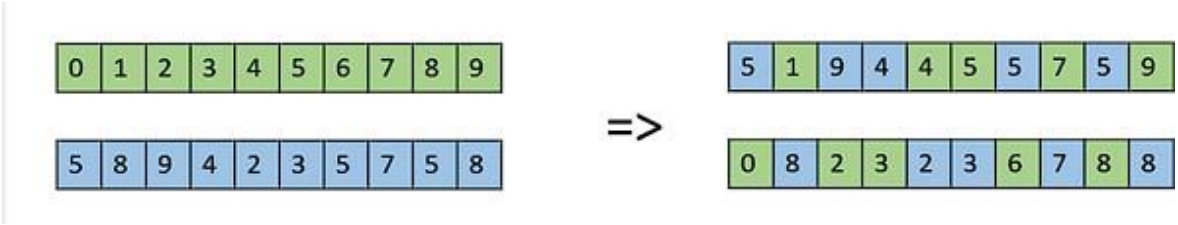
- One Point Crossover

Επιλέγεται ένα σημείο στο χρωμόσωμα και τα δύο μέρη των χρωμοσωμάτων που διασταυρώνονται ανταλλάσσονται αναπαράγοντας νέους απογόνους. Υπάρχει επίσης η Multi Point αναπαραγωγή όπου επιλέγονται πολλαπλά σημεία ανταλλαγής γονιδίων.



- Uniform Crossover

Σε αυτή την προσέγγιση δεν χωρίζουμε το χρωμόσωμα σε μέρη αλλά ανταλλάσσουμε τα γονίδια ξεχωριστά. Η αλλαγή ή όχι ενός γονιδίου εξαρτάται από μια πιθανότητα Pc.



- Whole Arithmetic Recombination

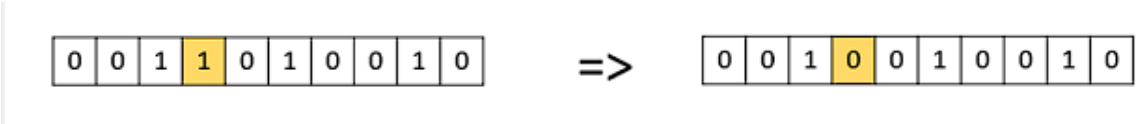
Τα γονίδια των παραγόμενων απογόνων υπολογίζονται σύμφωνα με τον τύπο : $Απόγονος1 = a * x + (1 - a) * y$ και $Απόγονος2 = a * x + (1 - a) * y$, με το a να είναι τυχαίος αριθμός. [17]

Mutation τεχνικές :

Ο τελευταίος μηχανισμός του γενετικού αλγορίθμου αποτελεί η τεχνική mutation κατά την οποία τα χρωμοσώματα μεταλλάσσονται και παραμορφώνονται σε σχέση με την αρχική τους κατάσταση. Η πιθανότητα να μεταλλαχθεί ένα χρωμόσωμα είναι συνήθως ίδια για όλα τα στοιχεία επομένως είναι απαραίτητο να εισαχθεί το mutation rate P_m πράγμα που θα αποτελεί την πιθανότητα να τροποποιηθεί ένα μεμονωμένο γονίδιο χρωμοσώματος. Παρακάτω παρουσιάζονται διάφοροι τρόποι μετάλλαξης.

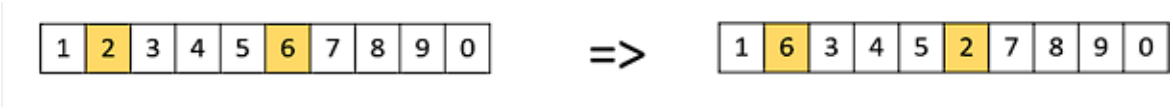
- Bit Flip Mutation

Εφόσον το χρωμόσωμα έχει γίνει encoded σε δυαδικό σύστημα το bit flip mutation θα αλλάξει ένα bit γονιδίου αλλάζοντας έτσι και το αποτέλεσμα άρα και το fitness.



- Swap Mutation

Επιλέγεται κάποια τυχαία γονίδια και ανταλλάσσονται οι τιμές μεταξύ τους.



- Inversion Mutation

Επιλέγονται πολλαπλά γονίδια και τα αντιστρέφουμε, μια ακόμη επίσης υλοποίηση είναι το κομμάτι γονιδίων που επιλέχθηκε να γίνει shuffle. [18]

3.2 Εφαρμογή γενετικού αλγορίθμου στο RBF δίκτυο

Η εφαρμογή γενετικού αλγορίθμου στο RBF δίκτυο μπορεί να εκφραστεί από την εξίσωση :

$$\min(\sum_{i=1}^{p}(y - f(x_i))^2)$$

Δηλαδή στόχος είναι να ελαχιστοποιηθεί η συνάρτηση κόστους [2.3] χρησιμοποιώντας μια τροποποιημένη εκδοχή γενετικού αλγορίθμου.

Το πρόβλημα βελτιστοποίησης ουσιαστικά χωρίζεται σε δύο φάσεις. Στην πρώτη, γίνεται μια εκτίμηση για βασικές παραμέτρους του RBF δικτύου χρησιμοποιώντας τον K-Means αλγόριθμο [2.1.1]. Στην δεύτερη φάση γίνεται χρήση του τροποποιημένου γενετικού αλγορίθμου ώστε να βρεθεί η βέλτιστη λύση για το εκάστοτε πρόβλημα. [19],[20]

3.2.1 Οριοθέτηση παραμέτρων δικτύου

Η προτεινόμενη έκδοση του αλγορίθμου χειρίζεται τον πληθυσμό ως χρωμοσώματα με διαστάσεις $(d + 1) * k$, όπου d είναι η διάσταση του εκάστοτε Dataset και k ο αριθμός των νευρώνων στο κρυμμένο επίπεδο. Η διάταξη του χρωμοσώματος φαίνεται στον παρακάτω πίνακα :

c_{11}	c_{12}	…	c_{1d}	σ	c_{21}	c_{22}	…	c_{2d}	σ	c_{k1}	c_{k2}	…	c_{kd}	σ

Το κάθε γονίδιο περιέχει κέντρα c_i και την διακύμανση σ τα οποία εκτιμώνται με την χρήση του K-Means αλγόριθμου. Έπειτα, αφού έχουν εκτιμηθεί οι παραπάνω παράμετροι πρέπει να βρεθούν τα όρια για να αρχικοποιηθεί ο αρχικός πληθυσμός. Τα διανύσματα $\vec{L}$, $\vec{R}$ περιέχουν τιμές από τις οποίες θα εξάγουμε το min,max οι οποίες τιμές θα χρησιμεύσουν ως τα όρια για τα γονίδια των χρωμοσωμάτων. [20]

Τα διανύσματα $\vec{L}$, $\vec{R}$ υπολογίζονται σύμφωνα με τον παρακάτω αλγόριθμο :

Αλγόριθμος Εκτίμησης Ορίων

1. generate random αριθμό F με F>1, m=0
2. Επανέλαβε
   - Για i από 1 έως k

   2.1 Επανέλαβε
   - Για i από 1 έως d
   - Κάνε :
     $L_m = -F * C_{ij}$
     $R_m = F * C_{ij}$
   - Κάνε $m = m + 1$

   2.4 Τέλος Επανάληψης

   2.5 Κάνε :
   $L_m = -F * \sigma$
   $R_m = F * \sigma$

   2.6 Κάνε $m = m + 1$

Αλγόριθμος 3. Εκτίμηση ορίων

3.3 Αλγόριθμος βελτιστοποίησης στο RBF

3.3.1 Αρχικοποίηση αρχικού πληθυσμού και παραμέτρων

Πριν εφαρμοστεί ο γενετικός αλγόριθμος πρέπει να γίνει initialize ο αρχικός πληθυσμός, πράγμα που προϋποθέτει να έχουμε διαβάσει το εκάστοτε dataset και να ξέρουμε τις διαστάσεις του προβλήματος, να ορίσουμε K αριθμό από νευρώνες για το κρυμμένο επίπεδο υπολογίζοντας τα κέντρα C_{ij} και την διακύμανση σ έτσι ώστε να μπορούμε να εκτελέσουμε τον αλγόριθμο εκτίμησης ορίων και να εκτιμήσουμε τον αρχικό πληθυσμό από χρωμοσώματα. Τέλος, πρέπει να οριστούν ο μέγιστος αριθμός από επαναλήψεις $\text{Iter}_{\max}$ όπου θα εκτελείτε ο γενετικός αλγόριθμος, το Selection Rate P_s $με$ $P_s \in [0,1]$, Mutation Rate P_m $με$ $P_m \in [0,1]$. Εφόσον έχει εκτιμηθεί ο αρχικός πληθυσμός θα έχουμε N αριθμό από χρωμοσώματα (σε μορφή όπως το figure 4) τα οποία σε κάθε επανάληψη πρέπει να αξιολογηθούν για να εκτελέσουμε τις ενέργειες του γενετικού αλγορίθμου. Σε κάθε επανάληψη μεταφέρουμε τα $(1 - P_s) * N$ καλύτερα χρωμοσώματα στην επόμενη γενιά χωρίς να συμμετέχουν στις ενέργειες διασταύρωσης, ενώ επιλέγονται τα $P_s * N$ χειρότερα χρωμοσώματα χρησιμοποιώντας την τεχνική Tournament Selection [3.1.3]. Για να γίνει επιλογή μεταξύ των χειρότερων και καλύτερων χρωμοσωμάτων πρέπει αυτά να αξιολογηθούν.

3.3.2 Αξιολόγηση χρωμοσωμάτων

Για την αξιολόγηση πρέπει πρώτα να αποκωδικοποιήσουμε τα χρωμοσώματα έτσι ώστε να εξάγουμε την διακύμανση σ και τα κέντρα $C_{ij.}$ Έπειτα, υπολογίζουμε τα βάρη σύμφωνα με $w = (A^T A)^{-1} * A^T y$. Είναι απαραίτητο σε κάθε evaluation που κάνουμε στο εκάστοτε χρωμόσωμα να υπολογίζουμε τα βάρη διότι για το συγκεκριμένο χρωμόσωμα με τα συγκεκριμένα κέντρα και βάρη υπολογίζεται το κόστος το οποίο πρέπει να ελαχιστοποιήσουμε [2.3].

3.3.3 Ενέργειες αναπαραγωγής

Τα χρωμοσώματα που επιλέχθηκαν μέσω του Tournament Selection θα συνδυαστούν με σκοπό την δημιουργία άλλων με χαμηλότερο κόστος. Αρχικά εκτελείται το crossover ως εξής :

Έστω i = 1 … d όπου d η διάσταση του συγκεκριμένου προβλήματος, για κάθε ζεύγος από επιλεγμένους γονείς εκτελούμε :

$$\text{α)} \qquad x_i = a_i + (1 - a_i) * y_i$$

$$\text{β)} \qquad y_i = a_i + (1 - a_i) * x_i$$

όπου α_i είναι τυχαίοι αριθμοί στο [-0.5 , 1.5].

Έπειτα το mutation εκτελείται ως εξής :

Έστω$[c_1, c_2, c_3 \ldots c_d]$ το επιλεγμένο χρωμόσωμα.

$$\acute{c}_i = \begin{cases} c_i + \Delta(iter, R_i - c_i) & \text{Εάν t>0.5} \\ c_i + \Delta(iter, ¢ - L_i) & \text{Αλλιώς} \end{cases}$$

Όπου το t είναι τυχαίος αριθμός στο [0,1] και η συνάρτηση Δ(iter,y) είναι :

$$\Delta(iter, y) = y * (1 - r^{(1 - \frac{iter}{iterMax})^b})$$

Όπου το r είναι τυχαίος αριθμός στο [0,1] και το b αποτελεί τον παράγοντα αλλαγής του c_i. [20]

3.3.4 Συνθήκη τερματισμού

Η συνθήκη τερματισμού είναι σημαντική για να μπορούμε να προσδιορίσουμε το πότε ο γενετικός αλγόριθμος θα πρέπει να τερματίσει. Αρχικά, ο αλγόριθμος είναι πιθανό να προχωρά αρκετά γρήγορα σε καλύτερες λύσεις σε κάθε επανάληψη. Σε ορισμένες περιπτώσεις όμως όταν ο αριθμός των επαναλήψεων πλησιάσει τον μέγιστο αριθμό ο αλγόριθμος μπορεί να συγκλίνει σε μία τιμή. Γι' αυτόν τον λόγο μια προσέγγιση είναι να υπάρχει ένας μετρητής ο οποίος θα αυξάνεται εάν δεν έχει παραχθεί μια καλύτερη λύση, όταν ο μετρητής αυτός φτάσει σε μια ορισμένη τιμή σημαίνει πως ο αλγόριθμος έχει συγκλίνει σε σταθερή τιμή και θα πρέπει να σταματήσει.

Στην συγκεκριμένη υλοποίηση ο αλγόριθμος σταματάει όταν οι επαναλήψεις φτάσουν τον μέγιστο αριθμό (IterMax), ο οποίος ελέγχεται από τον χρήστη.

Οπότε όταν η συνθήκη $Iter == IterMax - 1$ είναι αληθής ο αλγόριθμος σταματάει και πρέπει να γίνει ένα τελευταίο evaluation στα παραγόμενα χρωμοσώματα από τα οποία επιλέγεται το καλύτερο και παρουσιάζεται σαν βέλτιστη λύση. Προφανώς η παραπάνω τεχνική χρησιμοποιήθηκε για λόγους απλότητας μιας και η συγκεκριμένη συνθήκη τερματισμού μπορεί να επηρεάσει αρνητικά την ταχύτητα εκτέλεσης του αλγορίθμου.

4. Υλοποίηση σε C++ και δοκιμές σε πραγματικά προβλήματα

Όπως αναφέρεται και σε προηγούμενη ενότητα η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση όλων αυτών των μεθόδων είναι η C++. Μέρη του πηγαίου κώδικα κυρίως βασικών μεθόδων του δικτύου RBF αλλά και του γενετικού αλγορίθμου βρίσκονται στο παράρτημα ενώ μια πιο ολοκληρωμένη έκδοση του κώδικα βρίσκεται στο GitHub το οποίο χρησιμοποιήθηκε στην διάρκεια υλοποίησης της εργασίας με σκοπό να υπάρχει version control.

4.1 Πειραματικά αποτελέσματα και σύγκριση με το δίκτυο RBF

Θα συγκρίνουμε το output της συνάρτησης κόστους για το test set στα dataset που επιλέξαμε στο [2.4], αναδεικνύοντας το ποσοστό μείωσης μεταξύ του απλού RBF έναντι της χρήσης γενετικού αλγορίθμου. Θα χρησιμοποιήσουμε τα dataset που αναλύθηκαν στο [2.5] με τα πειραματικά αποτελέσματα να αντλούνται αφού τρέξουμε τον κάθε αλγόριθμο για όλα τα dataset 30 φορές και υπολογίσουμε τον μέσο όρο από το Test set.

Οι παράμετροι δικτύου που χρησιμοποιήθηκαν για τα πειράματα είναι οι εξής :

Παράμετρος	Τιμή
k	6
k_c	300
$Iter_{max}$	300
F	1.5
Selection rate	0.9
Mutation rate	0.05

Πίνακας 2 Παράμετροι δικτύου

Παρουσιάζονται τα αποτελέσματα του κλασικού RBF έναντι του γενετικού αλγορίθμου αλλά και το ποσοστό μείωσης του SSE.

Dataset	K-Means	Gen-RBF	Error decrease
Wine	4.497	3.097	31.1319%
WDBC	6.594	4.084	38.0649%
BK	1.319	1.286	2.5019%
Ionosphere	34.908	20.094	42.4373%
Iris	8.379	5.185	38.1191%

Πίνακας 3 Πειραματικά αποτελέσματα

Παράρτημα

Θα αναλύσουμε πως οι παραπάνω αλγόριθμοι και τεχνικές υλοποιήθηκαν σε C++. Αρχικά παρουσιάζονται οι υλοποιήσεις των συναρτήσεων κόστους δηλαδή το Sum squared test,train error αλλά και το classification error. Η υλοποίηση του Test, train error είναι ίδιες οπότε δεν θα παρουσιαστεί η μέθοδος TrainSumSquaredError μιας και η μόνη διαφορά μεταξύ των δύο είναι η αλλαγή του test σε train dataset.

```
double rbf::TestSumSquaredError() {
    Data summ;
    summ.resize(centers.size());
    double cnt;
    double sum = 0;
    for (int i = 0; i < test->rows(); i++) {
        summ.clear();
        cnt = 0;
        Data x = test->getX(i);
        for (int j = 0; j < centers.size(); j++)
            summ.push_back(gauss(x, centers[j], variance[j]));
        for (int k = 0; k < summ.size(); k++) {
            cnt += weights[k] * summ[k];
        }
        sum += pow(test->getY(i) - cnt, 2);
    }
    return 100 * sum / test->rows();
}
```

Το classification error υλοποιείται σύμφωνα με τον αλγόριθμο ο οποίος αναλύθηκε στην ενότητα 2.3.

```
void rbf::MinimumClassificationError() {
    Data summ;
    Matrix classes;
    classes.resize(Classes.size());
    for (int i = 0; i < classes.size(); i++)
        classes[i].resize(2);
    int random = 0 + (rand() % test->rows());
    Data xo = test->getX(random);
    double cnt, cnt1 = 0;
    for (int j = 0; j < centers.size(); j++)
        summ.push_back(gauss(xo, centers[j], variance[j]));
    for (int k = 0; k < summ.size(); k++) {
        cnt1 += weights[k] * summ[k];
    }
```

```
    Data min, max;
    min.resize(Classes.size());
    for (int i = 0; i < min.size(); i++)
        min[i] = 1e+100;
    max.resize(Classes.size());
    for (int i = 0; i < max.size(); i++)
        max[i] = 0;
    for (int i = 0; i < classes.size(); i++) {
        classes[i][0] = min[i];
        classes[i][1] = max[i];
    }
    int c = 0;
    for (int i = 0; i < test->rows(); i++) {
        summ.clear();
        cnt = 0;
        Data x = test->getX(i);
        double yy = test->getY(i);
        for (int n = 0; n < Classes.size(); n++) {
            if (isEqual(yy, Classes[n])) {
                for (int j = 0; j < centers.size(); j++)
                    summ.push_back(gauss(x, centers[j], variance[j]));
                for (int k = 0; k < summ.size(); k++) {
                    cnt += weights[k] * summ[k];
                }
                if (cnt < min[n]) {
                    min[n] = cnt;
                    classes[n][0] = cnt;
                } else if (cnt > max[n]) {
                    max[n] = cnt;
                    classes[n][1] = cnt;
                }
            }
        }
    }
    Data s;
    double cntt;
    double missed = 0;
    double category;
    for (int i = 0; i < test->rows(); i++) {
        s.clear();
        cntt = 0;
        Data x = test->getX(i);
        for (int j = 0; j < centers.size(); j++) {
            s.push_back(gauss(x, centers[j], variance[j]));
        }
        for (int k = 0; k < s.size(); k++) {
            cntt += weights[k] * s[k];
        }
```

```
        int index = i;
        for (int k = 0; k < classes.size(); k++) {
            if (classes[k][0] <= cntt && cntt <= classes[k][1]) {
                category = k;
                if (!(isEqual(test->getY(index), category))) {
                    missed++;
                    break;
                }
            }
        }
    }
    cout << "Minimum Classification error : " << 100 * missed / train->rows()
<< endl;
}
```

Για την εκπαίδευση του RBF όπως τονίζεται σε προηγούμενη ενότητα πρέπει να λυθεί το σύστημα $w = (A^T A)^{-1} * A^T y$, πράγμα που προϋποθέτει να υπάρχουν οι μέθοδοι πολλαπλασιασμού και υπολογισμού αντίστροφου και ανάστροφου πίνακα.

```
Matrix rbf::Multiply(Matrix &A, Matrix &B) {
    int n = A.size();
    int m = A[0].size();
    int p = B[0].size();
    int q = B.size();

    if (m != q) {

        cout << "Cannot multiply these 2 Matrices ,"
                " the number of columns in the first matrix must be equal to
the number of rows in the second matrix";
        exit(0);
    }
    Matrix c(n, vector<double>(p, 0));
    for (auto j = 0; j < p; ++j) {
        for (auto k = 0; k < m; ++k) {
            for (auto i = 0; i < n; ++i) {
                c[i][j] += A[i][k] * B[k][j];
            }
        }
    }

    return c;
}
```

```
Matrix rbf::Transpose(Matrix &A) {
    int rows = A.size();
    if (rows == 0) return {{}};
    int cols = A[0].size();
    Matrix r(cols, vector<double>(rows));
    for (int i = 0; i < rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            r[j][i] = A[i][j];
        }
    }
    return r;
}
```

```
Matrix rbf::Inverse(Matrix &mat) {

    auto height = mat.size();
    auto width = mat[0].size();
    Matrix result(height, Matrix::value_type(width));
    for (auto i = 0; i < width; ++i) {
        result[i][i] = 1;
    }
    for (auto j = 0; j < width; ++j) {
        auto maxRow = j;
        for (auto i = j; i < height; ++i) {
            maxRow = mat[i][j] > mat[maxRow][j] ? i : maxRow;
        }
        mat[j].swap(mat[maxRow]);
        result[j].swap(result[maxRow]);
        auto pivot = mat[j][j];
        auto &row1L = mat[j];
        auto &row1R = result[j];
        for (auto i = j + 1; i < height; ++i) {
            auto &row2L = mat[i];
            auto &row2R = result[i];
            auto temp = row2L[j];
            for (auto k = 0; k < width; ++k) {
                row2L[k] -= temp / pivot * row1L[k];
                row2R[k] -= temp / pivot * row1R[k];
            }
        }
        for (auto k = 0; k < width; ++k) {
            row1L[k] /= pivot;
            row1R[k] /= pivot;
        }
    }
```

```
    for (auto j = width - 1;; --j) {
        auto &row1L = mat[j];

        auto &row1R = result[j];
        for (auto i = 0; i < j; ++i) {
            auto &row2L =  mat[i];
            auto &row2R = result[i];
            auto temp = row2L[j];
            for (auto k = 0; k < width; ++k) {
                row2L[k] -= temp * row1L[k];
                row2R[k] -= temp * row1R[k];
            }
        }
        if (j == 0) break;
    }
    return result;
}
```

Εφόσον έχουν υλοποιηθεί οι παραπάνω μέθοδοι το μόνο που μένει για να λυθεί το σύστημα είναι ο υπολογισμός του ψευδοαντίστροφου πίνακα, ο οποίος αποτελεί μια αλληλουχία των παραπάνω μεθόδων.

```
Matrix rbf::Pseudo_Inverse(Matrix &A) {
    Matrix B = Transpose(A);
    Matrix V = Multiply(B, A);
    Matrix L = Inverse(V);
    Matrix C = Multiply(L, B);
    return C;
}
```

Έπειτα ο υπολογισμός των weights είναι απλός και αναλύεται παρακάτω.

```
void rbf::RbfTrain(vector<Data> c) {
    weights.clear();
    Matrix A;
    A.resize(count);
    for (int i = 0; i < A.size(); i++)
        A[i].resize(NumberOfWeights);

    for (int i = 0; i < train->rows(); i++) {
        Data x = train->getX(i);
        for (int j = 0; j < NumberOfWeights; j++)
            A[i][j] = gauss(x, c[j], variance[j]);
      }
```

```
    Matrix AA = Pseudo_Inverse(A);

    Matrix output;
    output.resize(train->rows());
    for (int i = 0; i < train->rows(); i++)
        output[i].push_back(train->getY(i));

    Matrix result = Multiply(AA,
                             output);

    for (int i = 0; i < result.size(); i++)
        weights[i] = result[i][0];
}
```

Στο επόμενο κομμάτι της ενότητας θα αναλύσουμε όλες τις ενέργειες του γενετικού αλγορίθμου, από την αρχικοποίηση του αρχικού πληθυσμού μέχρι και την έξοδο.

Αρχικά για να κάνουμε generate τυχαία τον αρχικό πληθυσμό πρέπει να υπολογίσουμε τα $\vec{L}$, $\vec{R}$ vectors σύμφωνα με τον αλγόριθμο εκτίμησης ορίων [3.2.1].

```
void rbf::setBounds() {

    Left.resize((train->cols() + 1) * getNumberOfWeights());
    Right.resize((train->cols() + 1) * getNumberOfWeights());
    int k = getNumberOfWeights();
    int m = 0;
    for (int i = 0; i < k; i++) {
        for (int j = 0; j < train->cols(); j++) {
            Left.at(m) = -F_constant * centers[i][j];
            Right.at(m) = F_constant * centers[i][j];
            m = m + 1;
        }
        Left.at(m) = -F_constant * variance[i];
        Right.at(m) = F_constant * variance[i];
        m = m + 1;
    }
}
```

Συνάμα, εφόσον έχουν εκτιμηθεί τα παραπάνω διανύσματα μπορούμε με μια μέθοδο να κάνουμε generate τυχαία chromosomes μέσα στο διάστημα $[\vec{L}, \vec{R}]$. Αυτό επιτυγχάνεται με μια μέθοδος setChromosomes ως εξής :

```
void rbf::setChromosomes(int n, double max, double min) {
    chromosomes.resize(n);
    for (int i = 0; i < chromosomes.size(); i++)
        chromosomes[i].resize((train->cols() + 1) * getNumberOfWeights());
    int m = 0;
    double lower_bound = min;
    double upper_bound = max;
    double a_random_double;
    std::uniform_real_distribution<double> unif(lower_bound, upper_bound);
    std::default_random_engine re;
    re.seed(time(NULL));
    for (int i = 0; i < n; i++) {
        m = 0;
        for (int j = 0; j < getNumberOfWeights(); j++) {
            for (int k = 0; k < train->cols(); k++) {
                a_random_double = unif(re);
                chromosomes[i][m] = a_random_double;
                m = m + 1;
            }
            chromosomes[i][m] = variance[0];
            m = m + 1;
        }
    }
}
```

Αφού έχουμε κάνει generate τον αρχικό πληθυσμό τα χρωμοσώματα θα έχουν την μορφή όπως το 3.2.1, με το κάτω όριο το minimum του $\vec{L}$ και το πάνω όριο το maximum του $\vec{R}$. Παρακάτω παρουσιάζονται τα όρια αυτά αλλά και η μορφή ενός χρωμοσώματος. Το Layout του χρωμοσώματος είναι με διάσταση d ίση με 4 γι' αυτό τον λόγο κάθε 4 κελιά επαναλαμβάνεται ο αριθμός 1.652993 που αποτελεί την διακύμανση σ για το συγκεκριμένο πρόβλημα.

```
Left Bound is : -11.1857
Right Bound is : 11.1857
[-10.558222| |10.256585| |-3.898803| |1.761896| |1.652993|
|-10.476864| |3.238729| |3.744298| |7.006339| |1.652993|
|-10.769180| |-4.724923| |-8.265581| |-10.125136| |1.652993|
|-9.869653| |1.695087| |1.571812| |8.716535| |1.652993|
|-9.733437| |-1.878442| |-6.082737| |-0.158146| |1.652993|
|1.186897| |5.094498| |1.322123| |-3.362340| |1.652993]
```

Εικόνα 8 Layout αρχικού χρωμοσώματος

Στο υπόλοιπο κομμάτι θα αναλύσουμε τα βήματα του γενετικού αλγορίθμου έως την έξοδο. Η λειτουργία του αλγορίθμου συμπεριλαμβάνεται σε μια μέθοδο runGen η οποία παίρνει ως παραμέτρους τον αριθμό N χρωμοσωμάτων που θα γίνει generate και τον μέγιστο αριθμό από επαναλήψεις. Σε κάθε iteration πρέπει να γίνει ένα αρχικό fitness evaluation έτσι ώστε να αξιολογηθούν. Κάτι τέτοιο επιτυγχάνεται ως εξής :

Αρχικά προσθέτουμε το αποτέλεσμα της συνάρτησης κόστους σε ένα διάνυσμα από το οποίο εξάγουμε τα $(1 - Ps) * N$ καλύτερα μέσω της συνάρτησης FindBestIndices με σκοπό να περάσουν στην επόμενη γενιά.

```
for (int i = 0; i < n; i++) { // Fitness Evaluation
          RbfTrain(convert(i));
          double sqe = genTrainSumSquaredError(convert(i));
          sum.push_back(sqe);
      }

      // 1) Pass GenSize chromosomes to the next generation unchanged
      min = findBestIndices(sum, GenSize);
```

Τα υπόλοιπα $Ps * N$ επιλέγονται σαν ζεύγη γονέων μέσω της μεθόδου Tournament Selection.

```
// 2) Tournament selection
      for (int v = 0; v < TournamentRate; v++) {
          tournamentIndexes.clear();
          tournamentSum.clear();
          int index = 0;
          for (int j = 0; j < 10; j++) {
              tournamentIndexes.push_back(distr(gen));
          }
          for (int k = 0; k < tournamentIndexes.size(); k++) {
```

```
                RbfTrain(convert(tournamentIndexes[k]));
            double sqe = genTrainSumSquaredError(convert(tournamentIndexes[k]));
                tournamentSum.push_back(sqe);
            }
            index = findBestIndices(tournamentSum, 1)[0];

            if (std::find(tournamentMax.begin(), tournamentMax.end(),
            tournamentIndexes[index]) != tournamentMax.end())

                index = findBestIndices(tournamentSum, 2)[1];

            tournamentMax.push_back(tournamentIndexes[index]);
        }
```

Μετά την επιλογή ακολουθούν οι μέθοδοι αναπαραγωγής crossover και mutation.

```
// 3) Perform Whole Arithmetic Recombination on the selected parents.
for (int i = 1; i < tournamentMax.size(); i += 2) {
    for (int j = 0; j < copychrom[0].size(); j++) {
        hold[i][j] = (copychrom[tournamentMax[i]][j] * a) + ((1 - a) *
copychrom[tournamentMax[i - 1]][j]);
        hold[i - 1][j] = (copychrom[tournamentMax[i - 1]][j] * a) + ((1 -
a) * copychrom[tournamentMax[i]][j]);
    }
}
```

```
 // 4) mutation procedure
        for (int i = 0; i < copychrom.size(); i++) {
            for (int j = 0; j < copychrom[i].size(); j++) {
                double tt = uni(r);
                if (tt >= 0.05) {
                    t = uni(r);
                    if (t > 0.5) {
                        double delta = Delta(iter, iterMax, decodedRight[j] -
                        copychrom[i][j], r_constant);
                        copychrom[i][j] = copychrom[i][j] + delta;
                    } else {
                        double delta = Delta(iter, iterMax, copychrom[i][j] -
                        decodedLeft[j], r_constant);
                        copychrom[i][j] = copychrom[i][j] - delta;
                    }
                }
            }
        }
```

Τέλος, εφόσον επιτευχθεί ο μέγιστος αριθμός επαναλήψεων πρέπει να γίνει το τελικό evaluation έτσι ώστε να κάνουμε output το καλύτερο chromosome. Παρακάτω παρουσιάζουμε ένα sample εξόδου.

```
   *** K-Means RBF Output ***

Mean Sum Squared Test Error : 14.2451
Mean Sum Squared Train Error : 13.6888
Minimum Classification error : 45.3333

 *** Genetic algorithm approach ***

enter number of chromosomes :100

Genetic algorithm output : 6.1041
Error decreased for : 55%

Process finished with exit code 0
```

Εικόνα 9 Έξοδος προγράμματος

Βιβλιογραφία

[1] Radial Basis Function Network - an overview | ScienceDirect Topics (2002). Available at: https://www.sciencedirect.com/topics/engineering/radial-basis-function-network.

[2] Radial Basis Function - an overview | ScienceDirect Topics (2019). Available at: https://www.sciencedirect.com/topics/veterinary-science-and-veterinary-medicine/radial-basis-function.

[3] Zhang, H. (2020) '*Cluster Analysis in Data mining'*, in Zhang, H., *Analyzing High-Dimensional Gene Expression and DNA Methylation Data with R*. 1st edn. Boca Raton, FL : CRC Press, 2020. | Series: Chapman & Hall/CRC mathematical and computational biology series: Chapman and Hall/CRC, pp. 57–99. Available at: https://doi.org/10.1201/9780429155192-5- http://user.engineering.uiowa.edu/~ie_155/lecture/K-means.pdf.

[4] Orr, M.J.L. (1996) *'Introduction to Radial Basis Function Networks*' , p. 67. Centre for Cognitive Science, University of Edinburgh, 2, Buccleuch Place, Edinburgh EH8 9LW, Scotland, Available at: https://faculty.cc.gatech.edu/~isbell/tutorials/rbf-intro.pdf.

[5] UCI Machine Learning Repository . Available at: https://archive.ics.uci.edu/ml/index.php.

[6] Schwenker, F., Kestler, H.A. and Palm, G. (2001) *'Three learning phases for radial-basis-function networks'*, Neural Networks, 14(4–5), pp. 439–458, Department of Neural Information Processing, University of Ulm, D-89069 Ulm, Germany. Available at: https://doi.org/10.1016/S0893-6080(01)00027-2.

[7] Feed Forward Neural Network (2019). *DeepAI*. Available at: https://deepai.org/machine-learning-glossary-and-terms/feed-forward-neural-network.

[8] An, S. (2021) *'Introduction to how an Multilayer Perceptron works but without complicated math'*, *CodeX*, 9 October. Available at: https://medium.com/codex/introduction-to-how-an-multilayer-perceptron-works-but-without-complicated-math-a423979897ac.

[9] Bullinaria, J.A. (2004) *'Radial Basis Function Networks: Introduction'*, p. 12. School of Computer Science, University of Birmingham, Birmingham B15 2TT, UK. Available at: https://www.cs.bham.ac.uk/~jxb/NN/l12.pdf.

[10] Beasley, D. (1993) *'An Overview of Genetic Algorithms : Part 1, Fundamentals'*, p. 16. Department of Computing Mathematics, University of Cardiff, Cardiff, CF24YN, UK, Available at: https://mat.uab.cat/~alseda/MasterOpt/Beasley93GA1.pdf.

[11] Busetti, F. (2001) *'Genetic Algorithms Overview'*, p. 14. School of Computer Science and Applied Mathematics, University of the Witwatersrand, Available at: https://www.researchgate.net/publication/2383112_Genetic_Algorithms_Overview.

[12] Carr, J. (2014) *'An Introduction to Genetic Algorithms'*, p. 40, Department of Mathematics and Statistics, Whitman College, Washington. Available at: https://www.whitman.edu/documents/academics/mathematics/2014/carrjk.pdf.

[13] Srivignesh R. (2021) *'Genetic Algorithms and its use-cases in Machine Learning'*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/06/genetic-algorithms-and-its-use-cases-in-machine-learning/.

[14] Kumar A. (2013) *'Encoding Schemes in Genetic Algorithm'*, p. 7, Department of Computer Science and Applications, Kurukshetra University, Kurukshetra. Available at: https://garph.co.uk/ijarie/mar2013/1.pdf.

[15] Mallawaarachchi, V. (2017) *'How to define a Fitness Function in a Genetic Algorithm?'*, *Medium*. Available at: https://towardsdatascience.com/how-to-define-a-fitness-function-in-a-genetic-algorithm-be572b9ea3b4.

[16] Genetic Algorithms - Parent Selection, *Tutorialspoint*. Available at: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_parent_selection.htm.

[17] Genetic Algorithms – Crossover, *Tutorialspoint*. Available at: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_crossover.htm.

[18] Genetic Algorithms – Mutation, *Tutorialspoint*. Available at: https://www.tutorialspoint.com/genetic_algorithms/genetic_algorithms_mutation.htm.

[19] Jia, W. *et al.* (2014) *'A New Optimized GA-RBF Neural Network Algorithm'*, Computational Intelligence and Neuroscience, 2014, School of Electrical and Information Engineering, Jiangsu University, Zhenjiang 212013, China. Available at: https://doi.org/10.1155/2014/982045.

[20] Tsoulos, I.G., Tzallas, A. and Karvounis, E. (2022) '*A Two-Phase Evolutionary Method to Train RBF Networks*', Applied Sciences, 12(5), p. 2439, Department of Informatics and Telecommunications, University of Ioannina, 45110 Ioannina, Greece. Available at: https://doi.org/10.3390/app12052439.