**ΕΠΛ 442: Μηχανική Μάθηση**

**ΕΡΓΑΣΙΑ 4: RBF Regression  
25/10/2023**

**Νικόλαος Θεοδώρου 1030496  
  
Εισαγωγή**

Σε αυτήν την αναφορά περιγράφονται οι λεπτομέρειες της εκπαίδευσης ενός νευρωνικού δικτύου για την επίλυση του προβλήματος πρόβλεψης δράσης χημικών ενώσεων. Περιλαμβάνονται οι χρησιμοποιηθείσες μεταβλητές, οι επιλογές που αφορούν τη δομή του νευρωνικού δικτύου και τα αποτελέσματα που προέκυψαν από την εκπαίδευση. Η εργασία τρέχει με την εντολή python3 Trainer.py και απαιτεί python 3.6 η νεότερη και τα πακέτα numpy, matplotlib, scikit-learn (sklearn) και έχει δοκιμαστεί σε περιβάλλον Linux/Debian.

**Παραμέτροι εισόδου στο parameters.txt**

**numHiddenLayerNeurons 10**

**numInputNeurons 53**

**numOutputNeurons 1**

**learningRates 0.1 0.2 0.1**

**sigmas 0.1**

**maxIterations 50**

**centresFile centers.txt**

**trainFile training.txt**

**testFile test.txt**

**dataFile selwood.txt**

**Προετοιμασία δεδομένων εισόδου**

Αρχικα τα δεδομένα θα πρέπει να κανονιικοποιηθούν πριν μπουν στο νευρωνικο δύκτιο. Χρησιμοποίησα την μέθοδο κανονικοποίσης MinMax για να κανονικοποιήσω κάθε στύλη των δεδομένων.  
Στη συνέχεια, μοίρασα τις πρώτες 21 εγγραφές των δεδομένων στο train set και τις υπόλοιπες 10 στο test set.

Η πιο πάνω διαδικασία γίνεται με τη βοήθεια του κώδικα που έγραψα στο αρχείο FileManager.py, με την κλάση DataHandler.

Αυτή η διαδικασία γίνεται αυτόματα με το τρέξιμο του script Trainer.py, πριν να εκκινήσει η εκπαίδευση.

**Μεταβλητές και Επιλογές για το Νευρωνικό Δίκτυο**

Για την επίλυση του προβλήματος της εργασίας υλοποίησα ένα νευρωνικό δύκτιο RBF, ακριβώς όπως περιγράφεται στο μάθημα. Δηλαδή, έβαλα 53 νευρώνες εισόδου όσο και το μέγεθος της εισόδου των δεδομένων και ένωσα τον κάθε νευρώνα εισόδου με όλους τους κρυφούς RBF νευρώνες του δυκτίου.

Αρχικά έγινε αρχικοποίηση των κέντρων χρησιμοποιώντας τον αλγοριθμό K-Means στα δεδομένα του train set, χρησιμοποιώντας την κλάση DataHandler, αφού τοποθετεί τα κέντρα σε ιδανικές συντεταγμένες ανάμεσα στα δεδομένα μας. Τα βάρη αρχικοποιήθκαν τυχαία.

Για τον έλεγχο της απόδοσης του δυκτίου, υπολόγισα instantaneous error μετά από κάθε εποχή.

Για την τοπολογία του δυκτίου δοκίμασα διάφορους αριθμούς από RBF Neurons στο hidden layer. Ο κάθε νευρώνας ενεργοποιήται με τη μη γραμμική Γκαουσιανή συνάρτηση.

Έφτασα στο συμπέρασμα πως με 10 νευρώνες μπορεί να λυθεί ικανοποιητικά το πρόβλημα.

Έγινε δοκιμή διάφορων παραμέτρων του νευρωνικου δυκτίου και για τα learning rates των συντεταγμένων των κεντρών, των βαρών και των σίγμα. Για το learning rate επέλεξα 0.1 0.2 0.1 αντίστοιχα, αφού δίνει ιδανική αναλογία χρόνου εκπαίδευσης και ακρίβειας, έχωντας υπόψη ότι τα κέντρα αρχικοποιήθηκαν σε ήδη βέλτιστες συντεταγμένες και δεν θα χρειαστεί πολλή εκπαίδευση.

Τέλος, αφού έγινε κανονικοποίηση των συντεταγμένων σε χώρο εύρους (0,1), για σιγμα επέλεξα 0.1, που δίνει μια κοντινή προσέγγιση με τα δεδομένα, αφού είναι και λίγα. Επέλεξα να δίνω μια τιμή σιγμα για όλα τα Kernels, για απλοποίηση της υλοποίησης.

Επίσης πρόσεξα πως με αυτό το ρυθμό μετά από 50 εποχές τα error rate τόσο στο training όσο και στο test set επιπεδόνεται. Το δύκτιο μαθαίνει τόσο γρήγορα, αφού έχω αρχικοποιήση σε βέλτιστες τοποθεσίες τα κέντρα.

Έχουμε λοιπόν τις πιο κάτω επιλογές που έγιναν για τη δομή του νευρωνικού δικτύου:

* Πρώτο Επίπεδο: 53 νευρώνες (53 είσοδοι)
* Κρυφό Επίπεδο: 10 RBF Neurons
* Ρυθμός Μάθησης κεντρών: 0.1
* Ρυθμός Μάθησης βαρών: 0.2
* Ρυθμός Μαθησης σίγμα: 0.1
* Αριθμός Επαναλήψεων (maxIterations): 50

**Ενδεικτικά δεδομένα αρχείου datafile.txt/training.txt/test.txt:**

Compound, Activity, ATCH1, ATCH2, ATCH3, ATCH4, ATCH5, ATCH6, ATCH7, ATCH8, ATCH9, ATCH10, DIPV\_X, DIPV\_Y, DIPV\_Z, DIPMOM, ESDL1, ESDL2, ESDL3, ESDL4, ESDL5, ESDL6, ESDL7, ESDL8, ESDL9, ESDL10, NSDL1, NSDL2, NSDL3, NSDL4, NSDL5, NSDL6, NSDL7, NSDL8, NSDL9, NSDL10, DWVOL, SURF\_A, MOFI\_X, MOFI\_Y, MOFI\_Z, PEAX\_X, PEAX\_Y, PEAX\_Z, MOL\_WT, S8\_1DX, S8\_1DY, S8\_1DZ, S8\_1CX, S8\_1CY, S8\_1CZ, LOGP, M\_PNT, SUM\_F, SUM\_R

K17,<-1\*,0.1685,3.86E-02,-8.40E-03,-0.101,3.50E-03,-0.241,-0.2524,0.4188,-0.4125,-0.4043,1.0017,-2.666,-3.28E-02,2.8481,-0.5455,-0.8034,-1.6953,-0.2885,-1.2909,-1.4224,-0.4025,-0.4663,-0.606,-0.3895,7.5333,4.1605,1.099,5.9683,4.0048,1.0283,1.2598,1.4602,0.4934,0.6803,250.4001,314.2267,574.2045,9189.6504,8711.1963,12.8093,0.9522,3.1573,264.3255,6.9012,9.7093,5.3613,5.2365,-5.0591,-0.5692,3.007,62,0.25,-0.23

D30,<-1\*,0.26,-0.1477,9.10E-02,-0.1629,0.1116,-0.2877,-0.2148,0.4257,-0.3958,-0.3246,3.1309,-1.3363,0.1914,3.4095,-0.6642,-0.946,-0.9866,-0.4211,-1.0243,-0.7004,-0.4143,-0.93,-0.7845,-0.3444,6.0402,3.4281,1.0804,4.0356,1.2915,1.6839,1.0902,1.8805,0.5986,1.0391,307.8,364.0264,1837.9072,15123.2236,13701.3262,13.6143,1.6903,4.7317,364.3583,9.434,10.5818,6.824,6.4758,-5.4216,-0.9107,3.686,178,0.67,0.16

**Έξοδος**

Η έξοδος γινεται στα αρχεία “*results/weights\_LR={learningRates}\_S={sigmas}\_EP={maxIterations}.txt*” με το κάθε ένα να έχει τις ανάλογες στήλες για epoch training/test error rate και την απεικόνηση .

**Ανάλυση Αποτελεσμάτων**

Σύμφωνα με τις πιο κάτω γραφική του results, οι προβλεπόμενες εξόδοι σχετίζονται σωστά με τις εισόδους, καταφέρνοντας να επιτύχει το επιθυμητό αποτέλεσμα για όλα τα παραδείγματα του συνόλου δεδομένων.

Τα αποτελέσματα δείχνουν ότι το νευρωνικό δίκτυο κατάφερε να μάθει να προβλέπει τις δραστηριότητες των χημικών ενώσεων. Οι μεταβλητές του δικτύου ενημερώθηκαν με βάση την αλγοριθμική διαδικασία που μάθαμε στο μάθημα. Δεν φαίνεται να κολλούμε σε τοπικά ελάχιστα.

**Γραφική Παράσταση**

Παρακάτω παρουσιάζονται γραφικές παραστάσεις για τα αποτελέσματα της εκπαίδευσης για το νευρωνικό δύκτιο που επέλεξα πιο πάνω:

A graph of a number of people

Description automatically generated with medium confidence

**Συμπεράσματα**

Η επιτυχής εκπαίδευση του νευρωνικού δικτύου να προσεγγίσει την ορθή πρόβλεψη, αποδεικνύει την αποτελεσματικότητα του δυκτίου RBF και της κατάλληλης δομής δικτύου.

**Κλάσεις που χρησιμοποιούνται στον Κώδικα**

Στον παρεχόμενο κώδικα, πολλές προσαρμοσμένες κλάσεις χρησιμοποιούνται για τη δημιουργία και τη διαχείριση ενός νευρωνικού δικτύου. Παρακάτω, θα συζητήσουμε καθεμία από αυτές τις κλάσεις και τους ρόλους τους στην υλοποίηση του νευρωνικού δικτύου.

### 1. Neuron

### The Neuron class represents an individual neuron in the Radial Basis Function (RBF) network. Neurons form the core units of the network, responsible for processing inputs, computing outputs, and propagating error during training. The class includes the following attributes and methods:

### Class Attributes:

### id (static): A static class-level attribute for assigning unique IDs to each neuron instance.

### Constructor - \_\_init\_\_(self, isBias=False, isOutput=False):

### Initializes a new Neuron instance.

### Parameters:

### isBias: Boolean indicating if the neuron is a bias neuron.

### isOutput: Boolean indicating if the neuron is an output neuron.

### Instance Attributes:

### id: Unique identifier for the neuron.

### connectedFromNeurons: List of neurons providing input to this neuron.

### connectedToNeurons: List of neurons receiving output from this neuron.

### output: Output value of the neuron (default is 1).

### delta: Delta value used during backpropagation.

### weights: Weights of connections to this neuron.

### weightsOld: Previous weights (not used in the provided code).

### isBias: Indicates if the neuron is a bias neuron.

### isOutput: Indicates if the neuron is an output neuron.

### Methods:

### addConnectedFromNeuron(neuron): Adds a neuron to the input list.

### addConnectedToNeuron(neuron): Adds a neuron to the output list.

### setOutput(output): Sets the neuron's output value.

### getOutput(): Returns the neuron's output value.

### setDelta(delta): Sets the neuron's delta value for backpropagation.

### getDelta(): Returns the neuron's delta value.

### setWeights(weights): Sets the connection weights for this neuron.

### getWeights(): Returns the connection weights.

### \_\_str\_\_(): Returns a string representation of the neuron, including ID, type, connections, output, delta, and weights.

### 2. RBFNeuralNetwork

### The RBFNeuralNetwork class represents a Radial Basis Function neural network, designed for regression tasks using Gaussian basis functions. The class includes attributes and methods for managing the network's operation:

### Attributes:

### parameters: Dictionary containing network parameters.

### centers: Center vectors for Gaussian functions.

### learningRate: Learning rate for the network.

### numInputNeurons: Number of input neurons.

### numOutputNeurons: Number of output neurons.

### numHiddenLayerNeurons: Number of hidden layer neurons.

### sigma: Standard deviations for Gaussian functions.

### inputNeurons: List of input neurons.

### outputNeurons: List of output neurons.

### hiddenLayerNeurons: List of hidden layer neurons.

### trainingError: List tracking training error.

### testingError: List tracking testing error.

### Methods:

### initializeNeurons(): Initializes neurons for all layers.

### initializeWeights(): Initializes connection weights.

### setInputs(inputs): Sets inputs for input neurons.

### calculateOutput(): Calculates network output.

### updateWeights(): Updates weights during backpropagation.

### train(data): Trains the network.

### test(data): Tests the network.

### calculateError(): Calculates network error.

### plotTrainingError(): Plots training error over epochs.

### plotTestingError(): Plots testing error over epochs.

### saveState(filename): Saves network state.

### loadState(filename): Loads network state.

### \_\_str\_\_(): Returns a string representation of the network.

### 3. DataHandler

### The DataHandler class is responsible for managing data-related operations, including reading, preprocessing, normalizing, splitting, and saving data:

### Methods:

### read\_data(): Reads data from a file.

### preprocess\_data(df): Preprocesses the data (e.g., normalizes features).

### split\_data(df, test\_size): Splits data into training and testing sets.

### save\_data(data, filename): Saves data to a file.

### 4. FileReader and FileWriter

### These classes handle reading and writing operations, particularly useful for reading training and testing datasets, network parameters, and writing error logs and performance metrics.