Πολυεπίπεδο Perceptron-Αλγόριθμος K-Means.

Τμήμα Μηχανικών Η/Υ Πανεπιστήμιο Ιωαννίνων 2021-2022

Μάθημα: Υπολογιστική Νοημοσύνη

Υπεύθυνος Καθηγητής: Λύκας Αριστείδης

Άσκηση 1:

Εντολή μεταγλώτισσης:

Το νευρωνικό δίκτυο με δύο κρυμμένα επίπεδα υλοποιήθηκε στο αρχείο p1.c και η μεταγλώττισή του γίνεται με την εξής εντολή:

```
gcc -o p1 p1.c -lm
```

Έπειτα, μπορούμε να τρέξουμε το εκτελέσιμο με την εντολή:

```
./p1
```

Το νευρωνικό δίκτυο με τρία κρυμμένα επίπεδα υλοποιήθηκε στο αρχείο p2.c και η μεταγλώττισή του γίνεται με την εξής εντολή:

```
gcc -o p2 p2.c -lm
```

Έπειτα, μπορούμε να τρέξουμε το εκτελέσιμο με την εντολή:

```
./p2
```

Τα αρχεία δεδομένων των training και test sets είναι τα αρχεία DatasetTest.txt, DatasetTrain.txt. Η εντολή "createDataset()" που τα δημιουργεί στην main του κώδικα έχει μπει σε σχόλια, ώστε να μη χρειάζεται να τρέχει κάθε φορά. Συμπεριλάβαμε τα αρχεία ώστε να εκτελείται ο κώδικας.

Κατασκευή Συνόλου Δεδομένων 1:

Σε πρώτο στάδιο της άσκησης έγινε ανάλυση του προβλήματος στο οποίο μας ζητείται να ανταπεξέλθουμε. Αρχικά, ξεκινήσαμε με την κατασκευή του συνόλου δεδομένων 1, καταλήξαμε πως πρέπει να δημιουργηθούν δύο αρχεία προκειμένου να αποθηκεύουμε τα τυχαία παραδείγματα x1,x2 στο διάστημα [-1,1] οι οποίες αποτελούν τις εισόδους στο νευρωνικό δίκτυο. Ανάλογα με το αν πληρούν ορισμένα κριτήρια που μας υποδείχθηκαν στην εκφώνηση οι συντεταγμένες τους, τα παραδείγματα αυτά ταξινομούνται σε κάποια ανάλογη κατηγορία. Η κατηγορία αυτή γράφεται στο ίδιο αρχείο σε δυαδική μορφή δίπλα από κάθε παράδειγμα. Καθώς οι πιθανές κατηγορίες είναι τέσσερις [C1,C2,C3,C4] οι αντίστοιχες κατηγορίες στο αρχείο είναι [(1,0,0,0),(0,1,0,0),(0,0,1,0),(0,0,0,1)]. Η προηγούμενη κατηγοριοποίηση έγινε με την σκέψη ότι αυτή η μορφή ανταποκρίνεται καλύτερα στις τιμές που θα έχω στο δίκτυο δηλαδή μεταξύ [0,1] και σε ευκολότερο έλεγχο στο τέλος μέσω της λογιστικής συνάρτησης. Τα παραπάνω επιτυγχάνονται μέσω της συνάρτησης `void createDataset();` στην οποία δημιουργούμε δύο αρχεία, ένα για την εκπαίδευση του δικτύου και ένα για τον έλεγχο, το καθένα με τέσσερις χιλιάδες παραδείγματα. Η συνάρτηση αυτή έχει κληθεί μία φορά στην αρχή της main(), δημιούργησε τα αρχεία "DatasetTrain.txt","DatasetTest.txt" και έπειτα έχει μπει σε κατάσταση σχολιασμού. Αν ο χρήστης επιθυμεί να εκπαιδεύσει το δίκτυο με κάποιο διαφορετικό σύνολο δεδομένων μπορεί να αφαιρέσει τα σχόλια στη main() και να δημιουργήσει εκ νέου αρχεία.

#Define :

Στο στάδιο αυτό, καθορίστηκε μέσω της εντολής define ποιά θα είναι τα επιθυμητά χαρακτηριστικά του δικτύου, ενώ ο χρήστης μπορεί να πραγματοποιήσει μεγάλες αλλαγές στο δίκτυο απλά αλλάζοντας αυτές τις μεταβλητές. Για παράδειγμα, να αλλάξει το ρυθμό μάθησης, των αριθμό των baches, τον αριθμό των νευρώνων σε κάθε κρυμμένο επίπεδο και άλλα.

```
#define trainValues 4000
#define testValues 4000
#define d 2 //arithmos eisodwn
#define K 4 //arithmos kathgoriwn
#define H1 10 //arithmos neyrwnwn prwto krymmeno epipedo
#define H2 10 //arithmos neyrwnwn deftero krymmeno epipedo
#define h 0.0005 //rythmos mathishs
#define whatFunc 0 //What function gia ton kathorismo ths relu an ==0 h tanh an ==1
#define errorDiff 10 //katwflh termatismou metaksi 2 epoxwn
#define B 4000
```

Φόρτωση του συνόλου εκπαίδευσης:

Προκειμένου να πάρουμε από τα αρχεία τα παραδείγματα που υλοποιήσαμε, έχει γίνει υλοποίηση της συνάρτησης `void createArrays();`. Μέσω αυτής δημιουργούνται οι πίνακες των παραδειγμάτων και των επιθυμητών κατηγοριών τόσο για την εκπαίδευση όσο και για τον έλεγχο μέσω της fscanf() απο τα ανάλογα αρχεία.

```
//Arrays for the training and testing of the network
float datasetArray[trainValues][d];
float T[trainValues][K];
float datasetArrayTest[testValues][d];//gia ton elegxo
float Ttest[testValues][K];//For testing the network will be used in Generalization
```

```
void createArrays(){
    int i,j;
    FILE *fp;
    fp=fopen("DatasetTrain.txt","r");
    for (i=0;i<trainValues;i++){
        for(j=0;j<d;j++){
            fscanf(fp,"%f",&datasetArray[i][j]);
        }
        for(j=0;j<K;j++){
            fscanf(fp,"%f",&T[i][j]);
        }
    }
    fclose(fp);
    fp=fopen("DatasetTest.txt","r");
    for (i=0;i<testValues;i++){
        for(j=0;j<d;j++){
            fscanf(fp,"%f",&datasetArrayTest[i][j]);
        }
        for(j=0;j<K;j++){
            fscanf(fp,"%f",&Ttest[i][j]);
        }
    }
    fclose(fp);
}
```

Καθορισμός αρχιτεκτονικής και καθολικών μεταβλητών:

Ύστερα από την κατανόηση της φύσης του προβλήματος, αλλά και κατά την διάρκεια του προγραμματισμού, προέκυψε η παρακάτω αρχιτεκτονική.

```
//Ylopoioun to Dataset S1.
void createDataset();
void printDataset();
void createArrays();
void setNoise();
```

```
void setWeights();//Dhnei mia fora sthn arxh tyxaies times sta barh
void forward_pass(float *x, int D, float *y, int k);//eythi perasma
void backprop(float *x, int D, float *t, int k);//opisthodromish
float computeInerProduct(float *x, float *y, int dmnsion);//ypologismos eswterikou ginomenou
```

```
void gradient_descent();
void setNewWeights();
void setZeros();
void computeGeneralization();//ypologizei thn genikeysh
int squareError();//ypologizei to tetragwniko sfalma
```

```
//Synarthseis pou xrhsimopoiountai prokeimenou
//na anaparasthsoume thn relu/tanh/logistikh
//kathws kai tis paragwgous afton
float relu(float u);
float d_relu(float u);
float tanhFunc(float u);
float d_tanh(float u);
float sigmoid(float u);
float d_sigmoid(float u);
```

Η συνάρτηση `void setNoise();`

Θα αλλάξει την επιθυμητή κατηγορία με πιθανότητα 10% του κάθε παραδείγματος με κάποια άλλη τυχαία κατηγορία. Η αλλαγή αυτή θα πραγματοποιηθεί στον καθολικό πίνακα T(που αναφέρθηκε στην προηγούμενη ενότητα) ο οποίος έχει τις επιθυμητές κατηγορίες των παραδειγμάτων.

Η συνάρτηση `void setWeights();`

Θα καθορίσει τις τιμές στα πρώτα τυχαία βάρη μεταξύ των συνδέσεων των νευρώνων καθώς και στις πολώσεις αυτών. Καθώς ο αριθμός των εισόδων θα είναι δύο ο αριθμός των νευρώνων του πρώτου κρυμμένου επιπέδου H1 και του δευτέρου αντίστοιχα H2 ενώ της εξόδου K. Οι τιμές αυτές αποθηκεύονται στους αντίστοιχους καθολικούς πίνακες πίνακες.

```
//weights and biases
float firstLevelWeights[H1][d];
float secondLevelWeights[H2][H1];
float thirdLevelWeights[K][H2];
float firstLevelPolwsh[H1];
float secondLevelPolwsh[H2];
float thirdLevelPolwsh[K];
```

●Η συνάρτηση `float relu(float u);`

Πραγματοποιεί την λειτουργία της relu.

```
float relu(float u){
    return fmaxf(0.0,u);
}
```

●Η συνάρτηση `float d_relu(float u);`

Είναι η παράγωγος της relu.

```
float d_relu(float u){
   if (u>0.0){
       return 1.0;
   }else{
       return 0.0;
   }
}
```

●Η συνάρτηση `float tanhFunc(float u);`
Είναι η συνάρτηση της υπερβολικής εφαπτομένης.

```
float tanhFunc(float u){
    return tanh(u);
}
```

●Η συνάρτηση `float d_tanh(float u);`
Αποτελεί την παράγωγο της υπερβολικής εφαπτομένης.

```
float d_tanh(float u){
    return (1.0-tanh(u)*tanh(u));
}
```

●Η συνάρτηση `float sigmoid(float u);`
Αποτελεί την λογιστική συνάρτηση.

```
float sigmoid(float u){
    return (1.0/(1.0+exp(-1*u)));
}
```

●Η συνάρτηση `float d_sigmoid(float u);`
Αποτελεί την παράγωγο της λογιστικής συνάρτησης.

```
float d_sigmoid(float u){
    return (sigmoid(u)*(1.0 - sigmoid(u)));
}
```

●Η συνάρτηση `float computeInerProduct(float *x, float *y, int dmnsion);`
Λαμβάνει ως είσοδο δύο διανύσματα χ και y καθώς και την διάστασή τους και υπολογίζει το εσωτερικό τους γινόμενο.

```
float computeInerProduct(float *x, float *y, int dmnsion){
    // compute iner product of two given vectors and return them.
    int i;
    float inerProduct=0.0;
    for (i=0;i<dmnsion;i++){
        inerProduct += x[i]*y[i];
    }
    return inerProduct;
}
```

forward_pass(float *x,int D,float *y,int K)

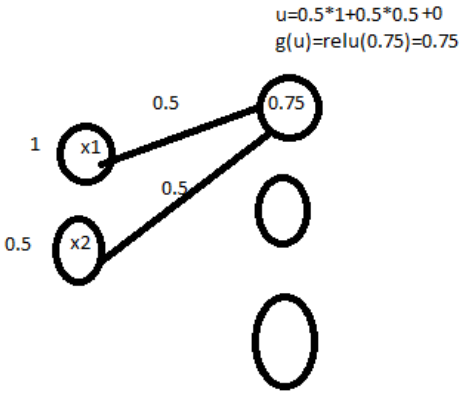
Η συνάρτηση αυτή υλοποιεί το ευθύ πέρασμα μέσα στο δίκτυο. Υπολογίζει το διάνυσμα εξόδου y διάστασης K δοθέντος το διάνυσμα εισόδου χ διάστασης D.

```
void forward_pass(float *x, int D, float *y, int k){
    //bres to dianisma y diastash k gia to dothen protypo x diastashs D.
    int i,j;
    for (i=0;i<H1;i++){//first level neurons
        firstLevelNeurons[i] = computeInerProduct(firstLevelWeights[i],x,D) + firstLevelPolwsh[i];//eswteriko ginomeno + polwsh.
        if (whatFunc==0){
            //analoga ti synarthsh exei dhlwthei sto define gia ta krymmena epipeda ypologizei to "g(u)".
            firstLevelNeuronsComplete[i]= relu(firstLevelNeurons[i]);
        }else{
            firstLevelNeuronsComplete[i]= tanh(firstLevelNeurons[i]);
        }

    }
    for (i=0;i<H2;i++){//second level neurons
        secondLevelNeurons[i]=computeInerProduct(secondLevelWeights[i],firstLevelNeuronsComplete,H1) + secondLevelPolwsh[i];
        if (whatFunc==0){
            secondLevelNeuronsComplete[i]= relu(secondLevelNeurons[i]);
        }else{
            secondLevelNeuronsComplete[i]= tanh(secondLevelNeurons[i]);
        }
    }
    for (i=0;i<K;i++){//exit level
        thirdLevelNeurons[i]=computeInerProduct(thirdLevelWeights[i],secondLevelNeuronsComplete,H2) + thirdLevelPolwsh[i];
        y[i]= sigmoid(thirdLevelNeurons[i]);//sto epipedo eksodou logistikh synarthsh kathw theloume times anamesa se 0-1.

    }
}
```

Παράδειγμα με συνάρτηση ενεργοποίησης την relu και πόλωση 0 :



Ο πίνακας firstLevelNeuron κρατάει για κάθε νευρώνα του πρώτου επιπέδου το u του παραπάνω παραδείγματος ενώ το το g(u) αποθηκεύεται στον firstNeuronComplete. Ενώ με την παράμετρο whatfunc που έχει γίνει define δηλώνεται αν θα χρησιμοποιηθεί η relu ή η tanh σαν συνάρτηση ενεργοποίησης. Το ίδιο συμβαίνει και για το δεύτερο επίπεδο, ενώ στο τρίτο δηλαδή στην έξοδο χρησιμοποιείται η λογιστική συνάρτηση καθώς οι κατηγορίες έχουν τιμές 0 ή 1. Οι παραπάνω πίνακες έχουν ορισθεί ως καθολικοί στην αρχή του προγράμματος.

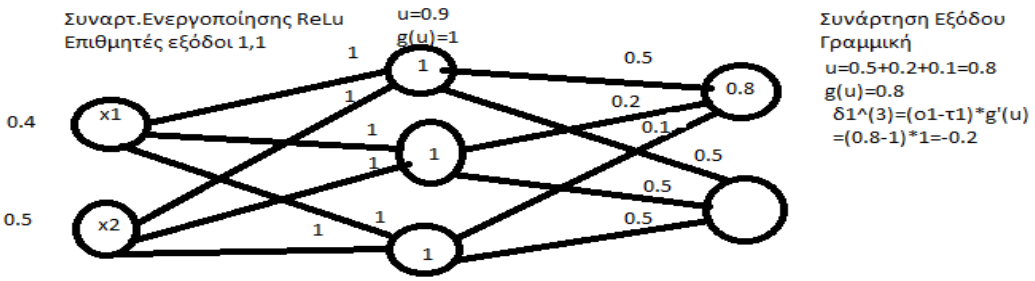
```
//Arrays for reflecting the Neurons of the network
//and the neurons after their activation fuction(...*Complete).
float firstLevelNeurons[H1];
float firstLevelNeuronsComplete[H1];
float secondLevelNeurons[H2];
float secondLevelNeuronsComplete[H2];
float thirdLevelNeurons[K];
float thirdLevelNeuronsComplete[K];
```

backprop(float *x,int d,float *t,int k)

Η συνάρτηση αυτή λαμβάνει τα διανύσματα x διάστασης d (είσοδος) και t διάστασης K (επιθυμητή έξοδος) και υπολογίζει τις παραγώγους του σφάλματος ως προς οποιαδήποτε παράμετρο (βάρος ή πόλωση) του δικτύου ενημερώνοντας τους αντίστοιχους πίνακες. Γίνεται αρχικά ευθύ πέρασμα και τα αποτελέσματα των εξόδων αποθηκεύονται στο πίνακα fwpassResult διάστακης K.Στη συνέχεια υπολογίζεται το σφάλμα προορισμού κάθε επιπέδου από το τέλος του δικτύου πρός το πρώτο επίπεδο(οπισθοδρόμηση).

```
void backprop(float *x, int D, float *t, int k){
    //opisthodromish
    int i,j;
    float fwpassResult[k];
    float sum;

    forward_pass(x,D,fwpassResult,k);
    //ypologise to sfalma "delta" se kathe epipedo
    //sfalma proorismou gia to teleftaio epipedo
    for (i=0;i<k;i++){
        deltaOut[i]=(fwpassResult[i]-t[i])*d_sigmoid(thirdLevelNeurons[i]);
    }
    //sfalma proorismou gia to deftero krymmeno epipedo
    for (i=0;i<H2;i++){
        sum=0.0;
        for (j=0;j<k;j++){
            sum += thirdLevelWeights[j][i]*deltaOut[j];
        }if (whatFunc==0){
            deltaSecondLevel[i] = sum*d_relu(secondLevelNeurons[i]);
        }else{
            deltaSecondLevel[i] = sum*d_tanh(secondLevelNeurons[i]);
        }
    }//sfalma proorismou gia to prwto krymmeno epipedo
    for (i=0;i<H1;i++){
        sum=0.0;
        for (j=0;j<H2;j++){
            sum += secondLevelWeights[j][i]*deltaSecondLevel[j];
        }if (whatFunc==0){
            deltaFirstLevel[i] = sum*d_relu(firstLevelNeurons[i]);
        }else{
            deltaFirstLevel[i] = sum*d_tanh(firstLevelNeurons[i]);
        }
    }
```



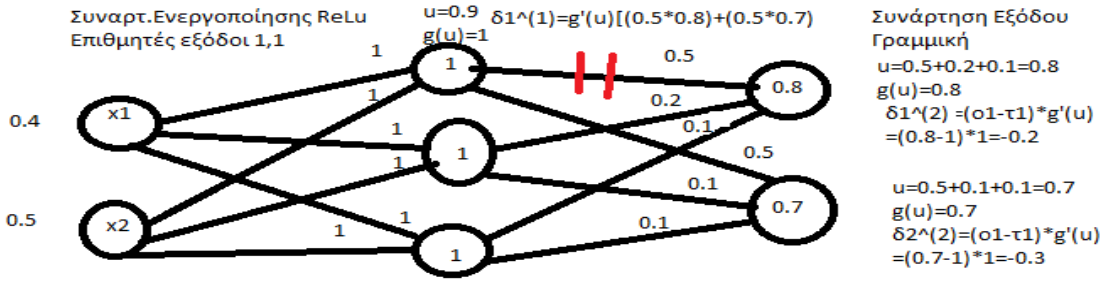
Σε τελικό στάδιο αφού έχουν υπολογιστεί όλα τα σφάλματα "δ" υπολογίζεται η μερική παράγωγος βάρους σύνδεσης (Σφάλμα προορισμού * έξοδος πηγής).

$$\frac{\partial E^n}{\partial w_{ij}^{(h)}} = \delta_i^{(h)} y_j^{(h-1)}$$

```
//sfalma proorismou gia kathe epipedo
float deltaOut[K];
float deltaSecondLevel[H2];
float deltaFirstLevel[H1];

//merikh metabolh kathe epipedou
float deh1[H1][d];
float deh1polwsh[H1];
float deh2[H2][H1];
float deh2polwsh[H2];
float deOut[K][H2];
float deOutPolwsh[K];
```

```
    for (i=0;i<H1;i++){
        for (j=0;j<d;j++){
            deh1[i][j]= deltaFirstLevel[i]*x[j];//gia epipedo H1
        }
        deh1polwsh[i]=deltaFirstLevel[i];
    }
    for (i=0;i<H2;i++){
        for(j=0;j<H1;j++){
            deh2[i][j] = deltaSecondLevel[i]*firstLevelNeuronsComplete[j];//gia to epipedo H2
        }
        deh2polwsh[i] = deltaSecondLevel[i];
    }
    for (i=0;i<k;i++){
        for (j=0;j<H2;j++){
            deOut[i][j] = deltaOut[i]*secondLevelNeuronsComplete[j];//Gia thn eksodo
        }
        deOutPolwsh[i]=deltaOut[i];
    }
}
```



Τώρα βρίσκω παράγωγο κάθε βάρους

dE^(n)/dW11^(2)= σφάλμα προορισμού *έξοδος πηγής
=-0.2*1= -0.2

Gradient_descent:

Ο αλγόριθμος gradient descent υλοποιείται μέσα από την συνάρτηση `gradient_descent()` η οποία καλεί πολλαπλές συναρτήσεις. Αρχικά, καλεί την συνάρτηση `setZeros();` και μέσω αυτής μηδενίζει τους πίνακες στους οποίους θα αποθηκευτεί άθροισμα των μερικών δηλαδή των συνολικό ρυθμό μεταβολής. Οι πίνακες αυτοί έχουν καθοριστεί εξίσου στην αρχή του προγράμματος ως καθολικοί.

```
void setZeros(){
    int i,j;
    //set every value to 0
    for(i=0;i<H1;i++){
        for(j=0;j<d;j++){
            deh1Complete[i][j]=0.0;
        }
        deh1polwshComplete[i]=0.0;
    }
    for(i=0;i<H2;i++){
        for(j=0;j<H1;j++){
            deh2Complete[i][j]=0.0;
        }
        deh2polwshComplete[i]=0.0;
    }
    for(i=0;i<K;i++){
        for(j=0;j<H2;j++){
            deOutComplete[i][j]=0.0;
        }
        deOutPolwshComplete[i]=0.0;
    }
}
```

```
//synolikos rythmos metabolhs
float deh1Complete[H1][d];
float deh1polwshComplete[H1];
float deh2Complete[H2][H1];
float deh2polwshComplete[H2];
float deOutComplete[K][H2];
float deOutPolwshComplete[K];
```

Στη συνέχεια ανάλογα με το αν η μεταβλητή B που έχει γίνει define στην αρχή, είναι ίση με το συνολικό αριθμό παραδειγμάτων, θα γίνει ομαδική ενημέρωση ή σειριακή ενημέρωση ή ενημέρωση σε batches.

```
void gradient_descent(){
    //ypologise tis metaboles mias epoxhs kai enhmerwse ta barh.
    int i,j,k;
    int batchCounter=0;
    setZeros();
    //Ypologise ton synoliko rythmo metabolhs(To athorisma twn merikwn) pou tha ypostei kathe neyrwnas sto telos mias epoxhs.
    if(B==trainValues){
        for (i=0;i<trainValues;i++){
            backprop(datasetArray[i],d,T[i],K);
            for (j=0;j<H1;j++){
                for (k=0;k<d;k++){
                    deh1Complete[j][k] += deh1[j][k];
                }
                deh1polwshComplete[j] += deh1polwsh[j];
            }
            for (j=0;j<H2;j++){
                for(k=0;k<H1;k++){
                    deh2Complete[j][k]+=deh2[j][k];
                }
                deh2polwshComplete[j]+=deh2polwsh[j];
            }
            for (j=0;j<K;j++){
                for (k=0;k<H2;k++){
                    deOutComplete[j][k]+=deOut[j][k];
                }
                deOutPolwshComplete[j]+= deOutPolwsh[j];
            }
        }
        setNewWeights();
```

Κάθε παράδειγμα συνόλου εκπαίδευσης καθώς και η επιθυμητή του κατηγορία δίνονται ως είσοδοι στην συνάρτηση backprop(), εκει γίνονται οι ενέργειες που αναφέρθηκαν παραπάνω

και συγκεκριμένα η ενημέρωση των μερικών σφαλμάτων. Για κάθε νευρώνα, σε κάθε κρυμμένο επίπεδο, καθώς και στους νευρώνες εξόδου υπολογίζεται ο Συνολικός ρυθμός μεταβολής με τον παρακάτω τρόπο.

$$\frac{\partial E}{\partial w_i} := \frac{\partial E}{\partial w_i} + \frac{\partial E^n}{\partial w_i}$$

Τέλος ενημερώνονται τα βάρη με βάση τον συνολικό ρυθμό μεταβολής που υπολογίστηκε καλώντας την συνάρτηση `setNewWeights();`

```
void setNewWeights(){
    int i,j,k;
    //bale tis synolikes metaboles sta brh
    for (i=0;i<H1;i++){
        for(j=0;j<d;j++){
            firstLevelWeights[i][j] -= (float)(h*deh1Complete[i][j]);
        }
    }
    for(j=0;j<H1;j++){
        firstLevelPolwsh[j] -= (float)(h*deh1polwshComplete[j]);
    }
    for (i=0;i<H2;i++){
        for(j=0;j<H1;j++){
            secondLevelWeights[i][j] -= (float)(h*deh2Complete[i][j]);
        }
    }
    for(j=0;j<H2;j++){
        secondLevelPolwsh[j] -= (float)(h*deh2polwshComplete[j]);
    }
    for (i=0;i<K;i++){
        for(j=0;j<H2;j++){
            thirdLevelWeights[i][j] -=  (float)(h*deOutComplete[i][j]);
        }
    }
    for(j=0;j<K;j++){
        thirdLevelPolwsh[j] -= (float)(h*deOutPolwshComplete[j]);
    }
}
```

Τα παραπάνω επαναλαμβάνονται και στην περίπτωση της σειριακής ή της ενημέρωσης σε baches εώς ότου δοθεί όλο το σύνολο των παραδειγμάτων.

```
counter=0;
while(counter!=trainValues){
    for(i=batchCounter;i<B;i++){
        backprop(datasetArray[i],d,T[i],K);
        for (j=0;j<H1;j++){
            for (k=0;k<d;k++){
                deh1Complete[j][k] += deh1[j][k];
            }
            deh1polwshComplete[j] += deh1polwsh[j];
        }
        for (j=0;j<H2;j++){
            for(k=0;k<H1;k++){
                deh2Complete[j][k]+=deh2[j][k];
            }
            deh2polwshComplete[j]+=deh2polwsh[j];
        }
        for (j=0;j<K;j++){
            for (k=0;k<H2;k++){
                deOutComplete[j][k]+=deOut[j][k];
            }
            deOutPolwshComplete[j]+= deOutPolwsh[j];
        }
    }
    setNewWeights();
    setZeros();
    batchCounter+=B;
    counter+=B;
```

Μετά τις παραπάνω διαδικασίες γίνεται η ολοκλήρωση μιάς εποχής και τυπώνεται το σφάλμα εκπαίδευσης του συνόλου σε εκείνη την εποχη μέσω της συνάρτησης.

`int squareError()`

```
int squareError(){
    //ypologise to  sfalma ekpaideyshs
    int i,j;
    float error=0.0;
    for (i=0;i<trainValues;i++){
        forward_pass(datasetArray[i],d,Ycomplete[i],K);
        for(j=0;j<K;j++){
            error+=(T[i][j]-Ycomplete[i][j])*(T[i][j]-Ycomplete[i][j]);
        }
    }
    return (error/2);
```

Για κάθε τιμή του συνόλου εκπαίδευσης γίνεται ευθύ πέρασμα προκειμένου να πάρουμε τιμές στους νευρώνες εξόδου στον πίνακα Ycomplete ο οποίος έχει οριστεί με define ως καθολικός στην αρχή του προγράμματος και έπειτα υπολογίζεται ο παρακάτω τύπος :

```
float Ycomplete[trainValues][K];//gia to | sfalma
```

$$\frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{p}(t_{nm}-o_m(x^n;w))^2$$

Τέλος, ελέγχεται η συνθήκη τερματισμού στην main και αν ικανοποιείται υπολογίζεται η γενίκευση και τερματίζει το πρόγραμμα. Η γενίκευση υπολογίζεται με την συνάρτηση `void computeGeneralization()`

Για κάθε στοιχείο στο σύνολο ελέγχου θα πραγματοποιηθεί ευθύ πέρασμα, το αποτέλεσμα του οποίου αποθηκεύεται στον πίνακα Ytest,εκει ελέγχουμε ποιός νευρώνας έδωσε την μέγιστη τιμή και σε ποια θέση βρίσκεται. Στη συνέχεια ελέγχεται ο πίνακας με την τη επιθυμητή έξοδο που έχουμε ορίσει για κάθε παράδειγμα Ttest ,σε ποιά θέση έχει τον άσσο. Εάν αυτές οι 2 θέσεις συμφωνούν, δηλαδή η θέση του max με την θέση του άσσου, το παράδειγμα γράφεται στο αρχείο positive.txt μαζί με την κατηγορία του και ο μετρητής των σωστών απαντήσεων προσαυξάνεται κατα 1, αλλιώς το παράδειγμα γράφεται στο αρχείο λανθασμένων απαντήσεων “Negative.txt” και ο μετρητής σωστών απαντήσεων δεν μεταβάλλεται.

```
void computeGeneralization(){
    int i,j,k,exitNeuron,answer=0;
    int pos =-3;
    float max =0.0;
    FILE *fp;
    for (i=0;i<testValues;i++){
        forward_pass(datasetArrayTest[i],d,Ytest[i],K);
        pos= -10;
        max=-10.0;
        exitNeuron=0;
        for(j=0;j<K;j++){
            if(Ytest[i][j]>max){
                max =Ytest[i][j];
                pos=j;
            }
        }
        for (k=0;k<K;k++){
            if(Ttest[i][k]==1){
                exitNeuron=k;
                break;
            }
        }
```

```
        if (pos==exitNeuron){
            fp=fopen("Positive.txt","a+");
            for(k=0;k<d;k++){
                fprintf(fp,"%1.3f \t",datasetArrayTest[i][k]);
            }
            fprintf(fp,"%d\n",pos);
            fclose(fp);
            answer+=1;
        }else{
            fp=fopen("Neggative.txt","a+");
            for(k=0;k<d;k++){
                fprintf(fp,"%1.3f \t",datasetArrayTest[i][k]);
            }fprintf(fp,"%d\n",pos);
            fclose(fp);
            answer+=0;
        }
    }
    printf("\nFound correct : %d Generalization : %2.2f%%\n",answer,(float)100.0*answer/testValues);

}
```

Τέλος η συνάρτηση main():

```
int main(){
    int i,j;
    int epoxes=0,endTrain=0;
    float sfalma=0.0,sfalmaBefore=0.0;
    srand(time(NULL));
    //createDataset();//Makes A new dataset (a new Train file and a new Test File)
    createArrays();
    setNoise();
    setWeights();
    do{
        gradient_descent();
        sfalmaBefore=sfalma;
        sfalma=squareError();
        printf("Epoxh : %d \t Sfalma: %f\n",epoxes,sfalma);
        epoxes++;
        if (epoxes>700){
            if (abs(sfalma-sfalmaBefore)<=errorDiff && sfalma<=400){
            break;
            }
        }
    }while(1);
    computeGeneralization();
    return 0;
}
```

Υλοποιεί όσα περιγράφονται παραπάνω. Κατα αναλογία όλα τα παραπάνω χρησιμοποιήθηκαν για την υλοποιήσει και με τρία κρυμμένα επίπεδα, στις συναρτήσεις backprop, forward_pass, gradient descent , newWeights, setNewWeights απλα χρησιμοποιήθηκε ένα επιπρόσθετο ” for loop “ προκειμένου να αναπαραστήσουμε το τρίτο κρυμμένο επίπεδο καθώς και ορίστηκαν οι καθολικοί πίνακές του.

Πορίσματα :

Ύστερα από πειράματα που έγιναν πάνω και στα δύο δίκτυα καταλήξαμε ότι παίρνουμε το καλύτερο δυνατό αποτέλεσμα με το πλήθος νευρώνων σε κάθε κρυμμένο επίπεδο να ισούται με 15. Όπως επίσης ότι, μέσω της tanh το σφάλμα μειώνεται πιό ομαλά χωρίς μεγάλα “spikes” ανάμεσα στις εποχές και φτάνει σε μικρότερες τιμές. Ακόμα, μέσω της tanh μπορέσαμε να πάρουμε τα ίδια και μεγαλύτερα ποσοστά γενίκευσης με την relu αλλά και σφάλματος εκπαίδευσης σε πολύ μικρότερο αριθμό εποχών.

Επίσης για να πετυχαίνουμε το βέλτιστο δυνατό αποτέλεσμα εκτός από το κατώφλι σφάλματος που έχει γίνει define ίσο με 2 στο πρόγραμμα μας έχουμε ορίσει και ένα άνω όριο σφάλματος πριν το τέλος των εποχών ανάλογα με τα χαρακτηριστικά του δικτύου. Τα αποτελέσματα που πήραμε για τα διάφορα παραδείγματα είναι τα ακόλουθα :

Για 3 κρυμμένα επίπεδα ΟΜΑΔΙΚΗ ΕΝΗΜΕΡΩΣΗ :

Γενίκευση 94.07% 8 νευρώνες 7.500 εποχές όριο σφάλματος<=350 Relu
Γενίκευση 95.20% 10 νευρώνες 20.000 εποχές όριο σφάλματος <=300 Relu
Γενίκευση 96.30% 12 νευρώνες 3.579 εποχές όριο σφάλματος <=300 Tanh
Γενίκευση 97.63% 12 νευρώνες 12.263 εποχές όριο σφάλματος <=250 Tanh
Γενίκευση 95.70% 12 νευρώνες 6.711 εποχές όριο σφάλματος <=300 Relu

Γενίκευση 94.45% 15 νευρώνες 5.500 εποχές όριο σφάλματος <=350 Relu
Γενίκευση 90.53% 15 νευρώνες 1.553 εποχές όριο σφάλματος <=400 Tanh

Για 3 κρυμμένα επίπεδα Mini Batch N/100 :

Με tanh:

Γενίκευση 97.75% Νευρώνες 8 Σφάλμα 275 Εποχές 700
Γενίκευση 96.07% Νευρώνες 10 Σφάλμα 284 Εποχές 700
Γενίκευση 97.63% Νευρώνες 12 Σφάλμα 280 Εποχές 806
Γενίκευση 97.82% Νευρώνες 12 Σφάλμα 250 Εποχές 1585
Γενίκευση 98.38% Νευρώνες 12 Σφάλμα 240 Εποχές 4308
Γενίκευση 98.50% Νευρώνες 12 Σφάλμα 220 Εποχές 20.000
Γενίκευση 98.75% Νευρώνες 15 Σφάλμα 240 Εποχές 8325
Γενίκευση 98.05% Νευρώνες 15 Σφάλμα 2100 Εποχές 12.160
Γενίκευση 97.82% Νευρώνες 15 Σφάλμα 250 Εποχές 2.115

Με relu:

Γενίκευση 94.88% Νευρώνες 8 Σφάλμα 300 Εποχές 7112
Γενίκευση 95.93% Νευρώνες 10 Σφάλμα 297 Εποχές 1665
Γενίκευση 95.10% Νευρώνες 12 Σφάλμα 297 Εποχές 3152
Γενίκευση 94.78% Νευρώνες 15 Σφάλμα 287 Εποχές 700
Γενίκευση 97.10% Νευρώνες 15 Σφάλμα 249 Εποχές 3884

Για 3 κρυμμένα επίπεδα Mini Batch N/10 :

Με tanh:

Γενίκευση 95.20% Νευρώνες 8 Σφάλμα 299 Εποχές 700
Γενίκευση 94.70% Νευρώνες 10 Σφάλμα 300 Εποχές 767
Γενίκευση 97.38% Νευρώνες 12 Σφάλμα 250 Εποχές 2638
Γενίκευση 97.05% Νευρώνες 15 Σφάλμα 250 Εποχές 1879

Με relu:

Γενίκευση 94.75% Νευρώνες 8 Σφάλμα 300 Εποχές 3500
Γενίκευση 95.45% Νευρώνες 10 Σφάλμα 298 Εποχές 1743
Γενίκευση 96.82% Νευρώνες 12 Σφάλμα 249 Εποχές 6773
Γενίκευση 97.30% Νευρώνες 15 Σφάλμα 247 Εποχές 3073

Για 2 κρυμμένα επίπεδα ΟΜΑΔΙΚΗ ΕΝΗΜΕΡΩΣΗ :

Η τιμή του σφάλματος στο δίκτυο με τα 2 κρυμμένα επίπεδα ήταν αισθητά μεγαλύτερη και για να φτάσουν το όριο τερματισμού σφάλματος που θεωρούμε σαν επιθυμητό ανάλογα με την αρχιτεκτονική του χρειαζόταν πολύ μεγαλύτερο πλήθος εποχών απο πρίν ενώ ο ρυθμός του μειωνόταν με πολύ μικρότερο βαθμό και κατ επεκταση πολυ μεγαλύτερη ώρα εκπαίδευσης .Για 8 νευρωνες στα κρυμμένα επίπεδα, το σφάλμα για να περάσει το όριο 500 χρειαζόταν κατα μέσο όρο 5.000-7.000 εποχες. Για κάθε ένα από τα πειράματα χρησιμοποιήθηκε κατώφλι μεταξύ δύο εποχών ίσο με 10. Κάποια από τα καλύτερα αποτελέσματα που είχαμε στα πειράματα :

Γενίκευση 94.03% 10 νευρώνες 3761 εποχές όριο σφάλματος<=400 Relu
Γενίκευση 93.05% 10 νευρώνες 2496 εποχές όριο σφάλματος<=400 Tanh
Γενίκευση 94.28% 12 νευρώνες 5750 εποχές όριο σφάλματος<=400 Relu

Γενίκευση 92.07% 12 νευρώνες 2731 εποχές όριο σφάλματος<=400 Tanh
Γενίκευση 91.57% 15 νευρώνες 2749 εποχές όριο σφάλματος<=400 Tanh

<u>Για 2 κρυμμένα επίπεδα Mini Batch N/100 :</u>

Με tanh:

Γενίκευση 94.72% Νευρώνες 8 Σφάλμα 350 Εποχές 12.941
Γενίκευση 96.35% Νευρώνες 10 Σφάλμα 350 Εποχές 1.744
Γενίκευση 96.43% Νευρώνες 12 Σφάλμα 300 Εποχές 1.980
Γενίκευση 98.72% Νευρώνες 12 Σφάλμα 250 Εποχές 10.942
Γενίκευση 98.45% Νευρώνες 15 Σφάλμα 250 Εποχές 4.981
Γενίκευση 98.70% Νευρώνες 15 Σφάλμα 240 Εποχές 6.558

Με relu:

Γενίκευση 93.60% Νευρώνες 8 Σφάλμα 400 Εποχές 3786
Γενίκευση 95.20% Νευρώνες 10 Σφάλμα 300 Εποχές 8949
Γενίκευση 95.30% Νευρώνες 12 Σφάλμα 300 Εποχές 1578
Γενίκευση 96.07% Νευρώνες 15 Σφάλμα 300 Εποχές 2775

<u>Για 2 κρυμμένα επίπεδα Mini Batch N/10 :</u>

Με tanh:

Γενίκευση 94.50% Νευρώνες 8 Σφάλμα 300 Εποχές 3946
Γενίκευση 96.72% Νευρώνες 10 Σφάλμα 280 Εποχές 7856
Γενίκευση 97.72% Νευρώνες 12 Σφάλμα 250 Εποχές 5638
Γενίκευση 98.40% Νευρώνες 15 Σφάλμα 240 Εποχές 4710

Με relu:

Γενίκευση 93.20% Νευρώνες 8 Σφάλμα 400 Εποχές 2783
Γενίκευση 94.95% Νευρώνες 10 Σφάλμα 350 Εποχές 1781
Γενίκευση 95.65% Νευρώνες 12 Σφάλμα 300 Εποχές 6784
Γενίκευση 96.15% Νευρώνες 15 Σφάλμα 280 Εποχές 2301

Συμπέρασμα:

Το όφελος που προκύπτει από το τρίτο κρυμμένο επίπεδο είναι πως το σφάλμα μειώνεται πολύ πιο γρήγορα και μπορούμε να φτάσουμε σε μικρότερο σφάλμα. Συνάρτηση ενεργοποίησης = Tanh. Αριθμος κρυμμένων Νευρώνων 15 σε κάθε επίπεδο. Παρατηρείται στην γραφική πως όλα τα σφάλματα "-" είναι στις πιο ακριανές θέσεις κάθε κατηγορίας δηλαδή μόνο εκεί λαμβάνονται λάθος αποφάσεις από το δίκτυο όταν ένα παράδειγμα είναι πολύ κοντά στο να ανήκει σε 2 κατηγορίες.