Εργασία στην Υπολογιστική Νοημοσύνη Ι

Αλεβιζοπούλου Αφροδίτη Α.Μ:3879 alevizopou@ceid.upatras.gr
Κωστής Αθανάσιος Α.Μ:3414 kostisa@ceid.upatras.gr

1 Προεπεξεργασία δεδομένων

Τα παρακάτω υποερωτήματα έχουν υλοποιηθεί στα αρχεία er1a.m, er1b.m, er1c.m και er1d.m αντίστοιχα.

1.1 Εισαγωγή δεδομένων στο MATLAB

Τα δεδομένα βρίσκονται στα αρχεία optdigits.tra και optdigits.tes και αποτελούν το σύνολο εκπαίδευσης και το σύνολο ελέγχου αντίστοιχα. Οι πρώτες 64 στήλες των αρχείων είναι οι είσοδοι και η τελευταία στήλη είναι ο κωδικός αριθμός της επιθυμητής εξόδου (τιμές από 0 έως 9 για τους δέκα χαρακτήρες που προσπαθούμε να αναγνωρίσουμε). Η εισαγωγή των δεδομένων στο MATLAB γίνεται μέσω της συνάρτησης *dlmread*, όπου διαβάζουμε τις τιμές των δεδομένων από τα δύο αρχεία και τα τοποθετούμε σε δύο ξεχωριστούς πίνακες. Train είναι ένα μητρώο διάστασης 3823×65 και περιέχει τα δεδομένα εκπαίδευσης και Test είναι ένα μητρώο διάστασης 1797×65 και περιέχει τα δεδομένα ελέγχου. Οι εντολές που εκτελέστηκαν σε MATLAB κώδικα είναι οι εξής:

```
% Read the ASCII-delimited numeric data file
% and return the data in output matrix.

Train = dlmread('optdigits.tra');
Test = dlmread('optdigits.tes');
```

1.2 Κανονικοποίηση δεδομένων

Αρχικά, δημιουργούμε δύο νέους πίνακες στους οποίους αποθηκεύουμε μόνο τα δεδομένα εισόδου του συνόλου δεδομένων, χρησιμοποιώντας τη συνάρτηση *dlmread*. Έτσι, προκύπτουν τα μητρώα trainInput, διάστασης 3823×64 και testInput, διάστασης 1797×64, τα περιεχόμενα των οποίων πρόκειται να κανονικοποιηθούν. Οι δύο νέοι πίνακες διαφέρουν μόνο κατά μία στήλη σε σχέση με τους αντίστοιχους αρχικούς, εφόσον διαβάζουμε και αποθηκεύουμε όλα τα δεδομένα των δύο αρχείων εκτός από την τελευταία στήλη (έξοδος).

Ο μετασχηματισμός μέγιστης και ελάχιστης τιμής αποτελεί γραμμικό μετασχηματισμό του αρχικού συνόλου δεδομένων, έτσι ώστε η νέα ελάχιστη τιμή να είναι -1 και η νέα μέγιστη τιμή να είναι +1. Η κανονικοποίηση των αριθμητικών τιμών των χαρακτηριστικών εισόδου του συνόλου δεδομένων στο διάστημα [-1 1] γίνεται μέσω της συνάρτησης *mapminmax*, εκτελώντας τις παρακάτω εντολές σε MATLAB κώδικα:

```
% Read only input data from file and return the data in output matrix.

trainInput = dlmread('optdigits.tra', ',', [0,0,3822,63]);
testInput = dlmread('optdigits.tes', ',', [0,0,1796,63]);

% Process matrices by mapping row minimum and maximum values to [-1 1]

Normalized_trainInput = mapminmax(trainInput);
Normalized_testInput = mapminmax(testInput);
```

1.3 Δημιουργία Διανύσματος Εξόδου

Η δημιουργία του διανύσματος εξόδου γίνεται με χρήση της κωδικής τιμής εξόδου από 0 έως 9 που παρέχεται για κάθε δείγμα (π.χ. 0→0000000001, 1→0000000010, 2→0000000100 κ.ο.κ).

Αρχικά, δημιουργούμε δύο νέους πίνακες, μέσω της συνάρτησης dlmread, στους οποίους αποθηκεύουμε μόνο τα δεδομένα εξόδου του συνόλου δεδομένων. Έτσι, προκύπτουν τα μητρώα trainOutput διάστασης 3823×1 και testOutput διάστασης 1797×1, με τιμές περιεχομένων από 0 έως 9. Στη συνέχεια δημιουργούμε, μέσω της συνάρτησης zeros, δύο μηδενικούς πίνακες, τους transformed_trainOutput διάστασης 3823×10 και transformed_testOutput διάστασης 1797×10 αντίστοιχα. Έπειτα, εξετάζουμε όλες τις γραμμές των μητρώων trainOutput και testOutput και ανάλογα με την τιμή του περιεχομένου τους (0-9) δημιουργείται το διάνυσμα εξόδου για κάθε σύνολο. Οι εντολές που εκτελέστηκαν είναι οι εξής:

```
% Read only output data from file and return the data in output matrix

trainOutput = dlmread('optdigits.tra', ',', 0, 64);
testOutput = dlmread('optdigits.tes', ',', 0, 64);

% Create array of all zeros

transformed_trainOutput = zeros(3823,10);
transformed_testOutput = zeros(1797,10);

% Create train output matrix

for k = 1:numel(trainOutput)
    v = trainOutput(k);
    transformed_trainOutput(k,10 - v) = 1;
end

% Create test output matrix

for k = 1:numel(testOutput)
    v = testOutput(k);
    transformed_testOutput(k,10 - v) = 1;
end
```

1.4 Αναδιάταξη εγγραφών κάθε συνόλου

Ως αναδιάταξη των εγγραφών ενός συνόλου θεωρούμε την αναδιάταξη των γραμμών του, αλλάζοντας τη σειρά με την οποία ο ταξινομητής θα δέχεται τα εκάστοτε δείγματα. Πριν προχωρήσουμε στην αναδιάταξη των εγγραφών, συνενώνουμε μέσω της συνάρτησης *horzcat* τους πίνακες εισόδου και εξόδου κάθε συνόλου σε έναν ενιαίο πίνακα για κάθε σύνολο. Έτσι, δημιουργούνται τα «προς αναδιάταξη» μητρώα newTrain και newTest, διάστασης 3823×74 και 1797×74 αντίστοιχα. Θα μπορούσαμε να μην προχωρήσουμε στην παραπάνω συνένωση και απλά να αναδιατάξουμε προσεκτικά τις εγγραφές των εκάστοτε μητρώων εισόδου και εξόδου, αρχικοποιώντας τη γεννήτρια τυχαίων αριθμών του MATLAB με τον ίδιο αριθμό πριν από κάθε αναδιάταξη που αφορά στο ίδιο σύνολο. Βασικός στόχος είναι να διατηρήσουμε τις σωστές αντιστοιχίσεις εξόδων σε πρότυπα εισόδου. Όταν το seed τεθεί σε μια συγκεκριμένη τιμή, ο αλγόριθμος παράγει πάντα την ίδια ακολουθία τυχαίων αριθμών, παράγοντας πανομοιότυπη επανάληψη της προηγούμενης αναδιάταξης.

Έπειτα, αρχικοποιούμε τη γεννήτρια του MATLAB με seed ίσο με το άθροισμα των αριθμών μητρώου των μελών της ομάδας και αναδιατάσουμε τις εγγραφές κάθε συνόλου με τυχαίο τρόπο, μέσω της συνάρτησης randperm. Έτσι, προκύπτουν τα μητρώα shuffled_newTrain και shuffled_newTest, που αντιστοιχούν στο σύνολο εκπαίδευσης και ελέγχου αντίστοιχα. Στη συνέχεια, το μητρώο shuffled_newTrain διαχωρίζεται στο μητρώο train_matrix, διάστασης 3823×64, που αποτελεί το μητρώο εκπαίδευσης του δικτύου μας και στο μητρώο target_matrix, διάστασης 3823×10, το οποίο αποτελεί το μητρώο επιθυμητής εξόδου για το σύνολο εκπαίδευσης. Οι εντολές που εκτελέστηκαν είναι οι εξής:

```
% Concatenate arrays horizontally

newTrain = horzcat(Normalized_trainInput, transformed_trainOutput);
newTest = horzcat(Normalized_testInput, transformed_testOutput);

% sets the "seed" of the pseudo-random number generator to 7293.
% Once the seed is set to a given value, the algorithm always produces
% the same sequence of random numbers. This is useful if we need to use
% the same random numbers more than once, or to produce identical runs
% of the same simulation.

rand('seed', 7293);

% Random permutation synolou ekpaideysis
shuffled_newTrain = newTrain(randperm(size(newTrain,1)),:);

% Random permutation synolou elegxou
shuffled_newTest = newTest(randperm(size(newTest,1)),:);

% Create train matrix
train_matrix = shuffled_newTrain(:,1:64);

% Create target matrix
target_matrix = shuffled_newTrain(:,65:74);

% Create test matrix
test_matrix = shuffled_newTest(:,1:64);

% Create test target matrix
test_target_matrix = shuffled_newTest(:,65:74);
```

Πριν τη δημιουργία του νευρωνικού δικτύου αντιστρέφουμε τόσο τα μητρώα εισόδου αλλά και αυτά της επιθυμητής εξόδου για τη σωστή εκτέλεση των εντολών δημιουργίας, εκπαίδευσης και προσομοίωσης του νευρωνικού δικτύου, όπως θα εξηγήσουμε παρακάτω. Εκτελούμε τις παρακάτω εντολές:

```
train_matrix = train_matrix'
target_matrix = target_matrix'
test_matrix = test_matrix'
test_target_matrix = test_target_matrix'
```

Πλέον τα μητρώα train_matrix, target_matrix, test_matrix και test_target_matrix έχουν διάσταση 64×3823, 10×3823, 64×1797 και 10×1797 αντίστοιχα.

Σημείωση: Για την υλοποίηση της άσκησης χρησιμοποιήθηκε η έκδοση R2013a του MATLAB.

2 Σχεδιασμός και Υλοποίηση Τεχνητού Νευρωνικού Δικτύου

Το νευρωνικό μας δίκτυο θα έχει ένα κρυφό επίπεδο και τόσους κρυφούς νευρώνες όσους προκύπτουν από τον παρακάτω κανόνα: H ≈ (I + O) / 2 , όπου H είναι ο αριθμός των νευρώνων του κρυφού επιπέδου, I ο αριθμός εισόδων και O ο αριθμός εξόδων του ΤΝΔ αντίστοιχα. Η παραπάνω πράξη, με δεδομένα τα I = 64 και O = 10, μας δίνει αποτέλεσμα 37. Έτσι, επιλέγουμε 37 κρυφούς νευρώνες για το δίκτυό μας.

Συνολικά, το δίκτυο που σχεδιάζουμε θα αποτελείται από:

- ένα επίπεδο εισόδου, το οποίο δεν έχει νευρώνες και απλά θα στέλνει τα σήματα εισόδου σε όλους τους νευρώνες του κρυφού επιπέδου.
- ένα κρυφό επίπεδο, το οποίο αποτελείται από 37 κρυφούς νευρώνες.
- ένα επίπεδο εξόδου, το οποίο αποτελείται από 10 νευρώνες.

Παρακάτω, περιγράφουμε αναλυτικά τη διαδικασία υλοποίησης του τεχνητού νευρωνικού δικτύου, δίνοντας τις απαραίτητες επεξηγήσεις για κάθε συνάρτηση που χρησιμοποιείται.

Η κατασκευή του δικτύου γίνεται με την παρακάτω εντολή:

```
% Create a feed-forward backpropagation network

net = newff(minmax(train_matrix),[37,10],{'tansig','purelin'},'traingdm');
```

- Το πρώτο όρισμα της συνάρτησης *newff* είναι ένα R×2 μητρώο το οποίο περιέχει την μέγιστη και την ελάχιστη τιμή για κάθε μία από τις R γραμμές του μητρώου εκπαίδευσης, καθορίζοντας με αυτό τον τρόπο το εύρος των τιμών των εισόδων που θα χρησιμοποιηθούν για τη δημιουργία του δικτύου. Η συνάρτηση *newff* θα κατασκευάσει ένα δίκτυο με R εισόδους. Η συνάρτηση *minmax* κατασκευάζει ακριβώς ένα τέτοιου τύπου μητρώο. Συγκεκριμένα, έχουμε 64 εισόδους στο δίκτυο, άρα θέλουμε το μητρώο που θα προκύψει από τη συνάρτηση *minmax* να έχει διάσταση 64×2. Αυτός είναι ο λόγος για τον οποίο αντιστρέφουμε το μητρώο train_matrix πριν τη δημιουργία του νευρώνα.
- Το δεύτερο όρισμα της συνάρτησης *newff* δηλώνει ότι θα έχουμε 37 κρυφούς νευρώνες στο κρυφό επίπεδο και 10 εξόδους στο επίπεδο εξόδου του δικτύου.
- Το τρίτο όρισμα είναι οι συναρτήσεις μεταφοράς του κάθε επιπέδου. Η προεπιλεγμένη συνάρτηση μεταφοράς για το κρυφό επίπεδο είναι η *tansig* και για το επίπεδο εξόδου η *purelin*.
- Το τέταρτο όρισμα είναι η συνάρτηση εκπαίδευσης του νευρωνικού δικτύου. Χρησιμοποιούμε τη συνάρτηση *traingdm*. Η συγκεκριμένη συνάρτηση εκπαίδευσης υλοποιεί τον αλγόριθμο της πίσω διάδοσης του σφάλματος με χρήση της παραμέτρου ορμής. Η παράμετρος ορμής παίρνει τιμές από 0 έως 1. Με χρήση της ορμής επιτυγχάνουμε αποφυγή των τοπικών ελαχίστων και απόσβεση των ταλαντώσεων γύρω από κάποια λύση. Αν όμως τεθεί ίση με 1, το δίκτυο δεν παρουσιάζει καμία ευαισθησία στις τοπικές κλίσεις της συνάρτησης σφάλματος και άρα, δε μαθαίνει σωστά.

Στη συνέχεια, αρχικοποιούμε τα βάρη και τις πολώσεις του νευρωνικού δικτύου που δημιουργήσαμε με κάποιες προεπιλεγμένες τιμές, μέσω της συνάρτησης *init*, εκτελώντας την παρακάτω εντολή:

```
% Initialize neural network

net = init(net);
```

Είμαστε έτοιμοι για την εκπαίδευση του δικτύου, εκτελώντας την παρακάτω εντολή:

```
% Train neural network

[net,tr] = train(net, train_matrix, target_matrix);
```

- Το πρώτο όρισμα της συνάρτησης *train* είναι το δίκτυο net.
- Το δεύτερο όρισμα της συνάρτησης *train* είναι το μητρώο εκπαίδευσης train_matrix διάστασης 64×3823.
- Το τρίτο όρισμα είναι το μητρώο «στόχος» (μητρώο επιθυμητής εξόδου) διάστασης 10×3823.

Η συνάρτηση *train* επιστρέφει το εκπαιδευμένο δίκτυο μαζί με το αρχείο εκπαίδευσης για τις εποχές και την απόδοση.

Το επόμενο βήμα μετά την εκπαίδευση είναι η εξομοίωση του δικτύου με το σύνολο εκπαίδευσης, προκειμένου να πάρουμε τις επιθυμητές μετρήσεις. Η εξομοίωση πραγματοποιείται εκτελώντας την παρακάτω εντολή:

```
% Simulate neural network

yTrain = sim(net, train_matrix);
```

- Το πρώτο όρισμα της συνάρτησης *sim* είναι το δίκτυο net.
- Το δεύτερο όρισμα είναι το υπό εξέταση μητρώο (το μητρώο εκπαίδευσης στην προκειμένη περίπτωση).

Η συνάρτηση *sim* επιστρέφει ένα μητρώο διάστασης $10 \times R$ (όπου R είναι ο αριθμός των στηλών του υπό εξέταση μητρώου) που αποτελεί την πραγματική έξοδο του συστήματός μας.

Ως συνάρτηση απόδοσης χρησιμοποιούμε τη συνάρτηση *mse*, η οποία υπολογίζει το μέσο τετραγωνικό σφάλμα, δηλαδή το μέσο όρο των σφαλμάτων υψωμένα στο τετράγωνο:

```
% mse (train set)

perfTrain = mse(net, target_matrix, yTrain);
```

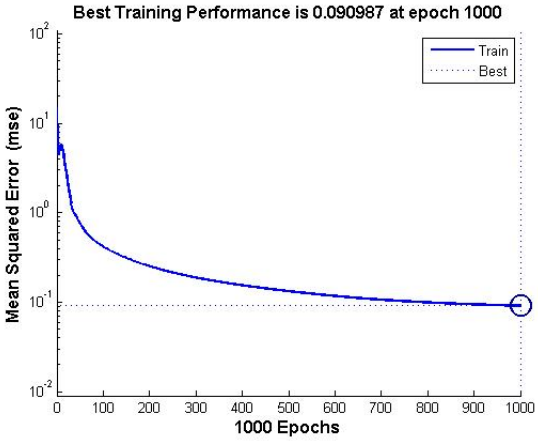
Για τον υπολογισμό της ακρίβειας πρόβλεψης υλοποιήσαμε τη συνάρτηση *percent*. Ο κώδικας της συνάρτησης *percent* και ο απαιτούμενος σχολιασμός παρατίθενται παρακάτω. Εκτελούμε το εξής:

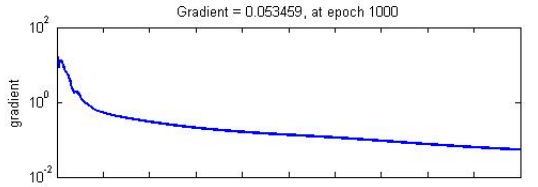
```
% Compute accuracy prediction (training set)

accurTrain = percent(yTrain, target_in_nums);
```

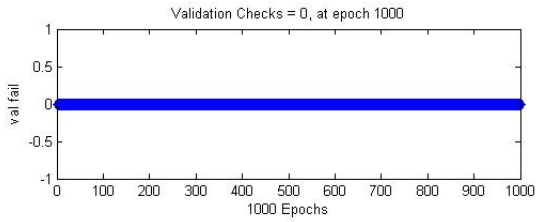
Ακριβώς με την ίδια λογική πραγματοποιούμε στη συνέχεια την εξομοίωση του δικτύου με το σύνολο ελέγχου, προκειμένου να πάρουμε τις αντίστοιχες μετρήσεις:

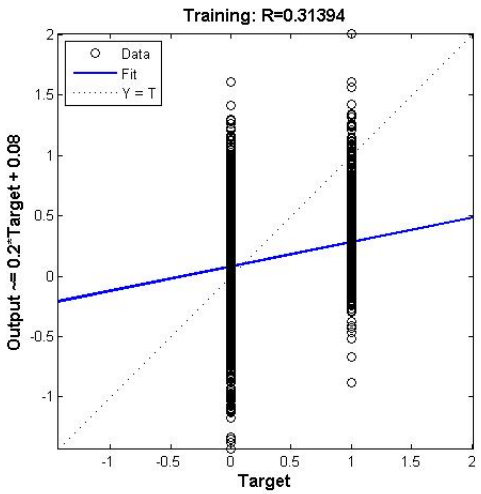
```
% Simulate neural network
yTest = sim(net, test_matrix);

% mse (test set)
perfTest = mse(net, test_target_matrix, yTest);

% Compute accuracy prediction (test set)
accurTest = percent(yTest, test_target_in_nums);
```

Παραθέτουμε τα screenshots που προκύπτουν από την εκπαίδευση του δικτύου:









Σύμφωνα με τα αποτελέσματα της εκτέλεσης, το μέσο τετραγωνικό σφάλμα και η ακρίβεια πρόβλεψης για το σύνολο εκπαίδευσης είναι 0.0910 και 0.4381. Αντίστοιχα, το μέσο τετραγωνικό σφάλμα και η ακρίβεια πρόβλεψης για το σύνολο ελέγχου είναι 0.0939 και 0.4079. Παρατηρούμε πως τα ποσοστά είναι σχετικά

χαμηλά. Επίσης, η ακρίβεια πρόβλεψης για το σύνολο ελέγχου είναι μικρότερη από αυτή του συνόλου εκπαίδευσης. Αυτό οφείλεται στο γεγονός ότι το δίκτυο ελέγχεται με δεδομένα εισόδου στα οποία δεν έχει εκπαιδευτεί.

Function *percent()* implementation

```
function accuracy = percent ( A, inNums )

counter = 0;
a = 0;

[maxVal maxInd] = max(A)

for a = 1:numel(maxInd)
    v = maxInd(a);
    maxVal(a) = 10 - v;
end

for i = 1:numel(inNums)
    if( inNums(i) == maxVal(i) )
        counter = counter + 1;
    end
end

accuracy = counter / a;

end
```

Πριν προχωρήσουμε στην πειραματική διαδικασία, θα αναλύσουμε τον τρόπο με τον οποίο ερμηνεύουμε τις εξόδους του κάθε νευρώνα αλλά και τη λογική της συνάρτησης *percent()* όσον αφορά στον υπολογισμό της ακρίβειας πρόβλεψης για το εκάστοτε σύνολο.

Θεωρούμε ότι το νευρωνικό δίκτυο ταξινομεί στον χαρακτήρα i (0-9) ανάλογα με το ποια από τις 10 εξόδους του έχει τη μεγαλύτερη τιμή. Για παράδειγμα, αν για ένα δείγμα έχουμε την ακόλουθη έξοδο:

Νευρώνας εξόδου 0: 0.1
Νευρώνας εξόδου 1: 0.5
Νευρώνας εξόδου 2: 0.7
Νευρώνας εξόδου 3: 0.6
Νευρώνας εξόδου 4: 0.2
Νευρώνας εξόδου 5: 0.1
Νευρώνας εξόδου 6: 0.15
Νευρώνας εξόδου 7: 0.3
Νευρώνας εξόδου 8: 0.6
Νευρώνας εξόδου 9: 0.1

Τότε, αυτό το δείγμα θα ταξινομηθεί στον χαρακτήρα 7 (χαρακτήρας Ή), αφού η μέγιστη έξοδος είναι αυτή του νευρώνα 2. Σύμφωνα με την εκφώνηση της άσκησης, το δείγμα αυτό ταξινομείται στον χαρακτήρα 2 (χαρακτήρας Ό). Ωστόσο, στο ερώτημα 1 της άσκησης αντιστοιχίζουμε τους νευρώνες εξόδου από αριστερά προς τα δεξιά στα ψηφία 9, 8, 7, 6, 5, 4, 3, 2, 1 και 0 (π.χ. 9→1000000000, 8→0100000000, 7→0010000000 κ.ο.κ). Αν λοιπόν φανταστούμε το διάνυσμα «γραμμή» που αντιστοιχεί στον χαρακτήρα 7 ως διάνυσμα «στήλη», συμπεραίνουμε ότι όταν η μέγιστη έξοδος είναι αυτή του νευρώνα εξόδου j, τότε το εν λόγω δείγμα πρέπει να ταξινομηθεί στον χαρακτήρα 9-j και όχι στο χαρακτήρα j.

Όσον αφορά στη συνάρτηση *percent()*, η βασική ιδέα είναι να συγκρίνουμε το διάνυσμα της επιθυμητής εξόδου (για το αντίστοιχο σύνολο) με το διάνυσμα της πραγματικής εξόδου και βάσει των ομοιοτήτων τους να προχωρήσουμε στον υπολογισμό της ακρίβειας πρόβλεψης για το εκάστοτε σύνολο. Για τη σύγκρισή τους, θεωρήσαμε λογικό να μετατρέψουμε τόσο το διάνυσμα της επιθυμητής εξόδου όσο και το διάνυσμα της πραγματικής εξόδου σε διανύσματα που θα περιλαμβάνουν τιμές από μηδέν έως εννέα και έπειτα να τα συγκρίνουμε.

Αναλυτικά, εντοπίζουμε αρχικά το νευρώνα εξόδου με τη μεγαλύτερη τιμή για κάθε ένα από τα δείγματα που δόθηκαν ως είσοδοι. Αυτό πραγματοποιείται μέσω της συνάρτησης *max*, η οποία μας επιστρέφει τη μέγιστη τιμή κάθε στήλης του πίνακα εξόδου καθώς και τη γραμμή του πίνακα στην οποία βρίσκεται η μέγιστη αυτή τιμή (οι τιμές των γραμμών είναι από ένα έως δέκα). Έτσι, δεδομένου ότι «αν η μέγιστη έξοδος είναι αυτή του νευρώνα εξόδου j, τότε το δείγμα ταξινομείται στον χαρακτήρα 9-j», συμπεραίνουμε σε ποιο χαρακτήρα (0-9) ταξινομήθηκε το εν λόγω δείγμα. Και επειδή, όπως αναφέραμε, οι γραμμές του πίνακα στο MATLAB έχουν αρίθμηση από ένα έως δέκα (και όχι απο μηδέν έως εννέα),«αν η μέγιστη έξοδος είναι αυτή της γραμμής j, τότε το δείγμα ταξινομείται στον χαρακτήρα 10-j». Αυτή η διαδικασία επαναλαμβάνεται για όλα τα δείγματα εισόδου, ώστε στο τέλος να δημιουργήσουμε ένα διάνυσμα που θα περιέχει το χαρακτήρα στον οποίο ταξινομήθηκε το κάθε δείγμα εισόδου.

Τη μετατροπή του διανύσματος «στόχου» την πραγματοποιούμε εκτελώντας τον παρακάτω κώδικα. Αρχικοποιούμε τη γεννήτρια του MATLAB θέτοντας το seed στην ίδια τιμή με πριν. Έτσι, οι αριθμοί αναδιατάσσονται με τον ίδιο ακριβώς τρόπο που πραγματοποιήθηκαν και οι προηγούμενες αναδιατάξεις, διατηρώντας τις σωστές αντιστοιχίσεις εξόδων σε πρότυπα εισόδου:

```
% Target matrix (0-9)

rand('seed', 7293);
target_in_nums = trainOutput(randperm(size(trainOutput,1)),:);
target_in_nums = target_in_nums'

% Test Target matrix (0-9)

test_target_in_nums = testOutput(randperm(size(testOutput,1)),:);
test_target_in_nums = test_target_in_nums'
```

Πλέον, μπορούμε να προχωρήσουμε στη σύγκριση των διανυσμάτων επιθυμητής και πραγματικής εξόδου. Έτσι, για κάθε κοινό χαρακτήρα που συναντάμε σε κοινή θέση στα δύο διανύσματα, αυξάνουμε έναν μετρητή κατά ένα, θεωρώντας ότι το δίκτυο προέβλεψε σωστά το συγκεκριμένο χαρακτήρα.

Τέλος, υπολογίζουμε την ακρίβεια πρόβλεψης διαιρώντας το σύνολο των «σωστά προβλεφθέντων χαρακτήρων» με το συνολικό αριθμό δειγμάτων που δόθηκαν ως είσοδοι στο δίκτυο.

3 Πειραματικά Αποτελέσματα και Συμπεράσματα

3.1 Έναρξη πειραμάτων & Συμπλήρωση πίνακα

Συμπληρώνουμε τον πίνακα του ερωτήματος 3 επαναλαμβάνοντας την παρακάτω ακολουθία βημάτων 5 φορές για τον εκάστοτε συνδυασμό παραμέτρου μάθησης και ορμής που ζητείται και παρουσιάζοντας το μέσο όρο των αποδόσεων. Σαν συνάρτηση ενεργοποίησης χρησιμοποιούμε τη λογιστική σε όλους τους νευρώνες του επιπέδου εξόδου και του κρυφού επιπέδου. Ο πλήρης κώδικας παρατίθεται τόσο στο παράρτημα στο τέλος του κειμένου, όσο και στο αντίστοιχο προς εκτέλεση αρχείο:

```
% Create a feed-forward backpropagation network

net = newff(minmax(train_matrix),[37,10],{'logsig','logsig'},'traingdm');
net = init(net);

net.trainParam.epochs = 100;
net.trainParam.max_fail = 10;
net.trainParam.min_grad = 1e-10;

net.trainParam.lr = 0.01;           % 0.01 or 0.1
net.trainParam.mc = 0.9;            % 0.9 or 0.6 or 0.1

[net,tr] = train(net, train_matrix, target_matrix);

yTrain = sim(net, train_matrix);

perfTrain = mse(net, target_matrix, yTrain);

accurTrain = percent(yTrain, target_in_nums);

yTest = sim(net, test_matrix);

perfTest = mse(net, test_target_matrix, yTest);

accurTest = percent(yTest, test_target_in_nums);
```

Οι τιμές του πίνακα αποτελούν το μέσο όρο των 5 επαναλήψεων που πραγματοποιήσαμε:

A/A	lr	mc	Epochs	MSE (train)	MSE (test)	accuracy (train)	accuracy (test)
1	0.01	0.9	100	0.2388	0.2397	0.1023	0.1032
2	0.01	0.6	100	0.2390	0.2399	0.1024	0.1033
3	0.01	0.1	100	0.2391	0.2399	0.1024	0.1034
4	0.1	0.9	100	0.1010	0.1011	0.1872	0.1854
5	0.1	0.6	100	0.0981	0.0982	0.1923	0.1914
6	0.1	0.1	100	0.0976	0.0977	0.1936	0.1938

Ας αναλύσουμε τα αποτελέσματα που προέκυψαν. Παρατηρούμε πως η απόδοση των πειραμάτων είναι χαμηλή. Αυτό κατά βάση οφείλεται στο γεγονός ότι το δίκτυο εκπαιδεύτηκε για ένα σχετικά μικρό χρονικό διάστημα (100 εποχές).

Επίσης, παρατηρούμε πως υπάρχουν διαφοροποιήσεις στην απόδοση με χρήση διαφορετικών τιμών παραμέτρων. Ο αλγόριθμος πίσω διάδοσης δίνει μια προσέγγιση της φθίνουσας καμπύλης προς το ελάχιστο στο υπερεπίπεδο του χώρου των βαρών, χρησιμοποιώντας της μέθοδο της απότομης μεταβολής. Επειδή η μέθοδος της μάθησης με διόρθωση σφάλματος συμπεριφέρεται σαν σύστημα κλειστής αναδρομής, είναι εμφανές ότι η τιμή της παραμέτρου μάθησης πρέπει να επιλεγεί με προσοχή, ώστε να εξασφαλιστεί η σταθερότητα της διαδικασίας. Αυτό συμβαίνει λόγω του ότι η παράμετρος μάθησης έχει μεγάλη επίδραση στην απόδοση της μεθόδου και επιδρά, όχι μόνο στην ταχύτητα σύγκλισης της μάθησης, αλλά και στην ίδια την κατάληξή της. Έτσι, όσον αφορά στην παράμετρο μάθησης, γενικά ισχύει ότι όσο μικρότερη είναι η τιμή της, τόσο μικρότερες είναι οι διορθώσεις που γίνονται στα βάρη σε κάθε βήμα και τόσο πιο ομαλή θα είναι η φθίνουσα καμπύλη στο υπερεπίπεδο, δηλαδή θα έχουμε πιο σταθερή σύγκλιση. Όμως, όλες οι βελτιώσεις έχουν σαν

αντίτιμο πιο αργό ρυθμό μάθησης. Δεδομένου ότι ο αλγόριθμος πίσω διάδοσης τρέχει μόνο για 100 εποχές στα πειράματά μας, χρειαζόμαστε μεγαλύτερη τιμή της παραμέτρου μάθησης, έτσι ώστε να επιτυγχάνεται η γρήγορη πορεία της μάθησης, η ελάττωση του αριθμού των επαναλήψεων του αλγορίθμου (άρα, ταχύτερη σύγκλιση) και καλύτερα αποτελέσματα.

Με χρήση της παραμέτρου της ορμής επιτυγχάνουμε αποφυγή των τοπικών ελαχίστων και απόσβεση των ταλαντώσεων γύρω από κάποια λύση. Η παράμετρος ορμής παίρνει τιμές από 0 έως 1. Αν όμως τεθεί ίση με 1, το δίκτυο δεν παρουσιάζει καμία ευαισθησία στις τοπικές κλίσεις της συνάρτησης σφάλματος και άρα, δε μαθαίνει σωστά. Η παράμετρος της ορμής συνήθως επιταχύνει τη σύγκλιση.

Σχετικά με την ικανότητα γενίκευσης, παρατηρούμε ότι είναι καλύτερη για τη μικρότερη κάθε φορά τιμή της παραμέτρου ορμής.

3.2 Έναρξη πειραμάτων με διπλάσιο αριθμό κρυφών νευρώνων

Εκτελούμε τα πειράματά μας για το βέλτιστο σύνολο παραμέτρων που εντοπίσαμε στο προηγούμενο ερώτημα, δηλαδή για lr = 0.1 και mc = 0.1 . Η παρακάτω ακολουθία βημάτων επαναλαμβάνεται 5 φορές. Ο πλήρης κώδικας παρατίθεται τόσο στο παράρτημα στο τέλος του κειμένου, όσο και στο αντίστοιχο προς εκτέλεση αρχείο:

```
net = newff(minmax(train_matrix),[74,10],{'logsig','logsig'},'traingdm');

net = init(net);

net.trainParam.epochs = 100;
net.trainParam.max_fail = 10;
net.trainParam.min_grad = 1e-10;

net.trainParam.lr = 0.1;
net.trainParam.mc = 0.1;

[net,tr] = train(net, train_matrix, target_matrix);

yTrain = sim(net, train_matrix);

perfTrain = mse(net, target_matrix, yTrain);

accurTrain = percent(yTrain, target_in_nums);

yTest = sim(net, test_matrix);

perfTest = mse(net, test_target_matrix, yTest);

accurTest = percent(yTest, test_target_in_nums);
```

Παραθέτουμε τον πίνακα που προκύπτει με χρήση 74 κρυφών νευρώνων:

A/A	lr	mc	Epochs	MSE (train)	MSE (test)	accurancy (train)	accurancy (test)
1	0.1	0.1	100	0.1358	0.1361	0.1956	0.1945

Παρατηρούμε ότι παρά την αύξηση του αριθμού των κρυφών νευρώνων τα αποτελέσματα που παίρνουμε είναι πολύ κοντά στα προηγούμενα. Αποτέλεσμα το οποίο εκ πρώτης όψεως δεν φαίνεται να συμφωνεί με τη θεωρία. Για το λόγο αυτό επαναλαμβάνουμε τα πειράματα για 37 και 74 νευρώνες, αλλά αυτή τη φορά χρησιμοποιούμε μεγαλύτερο αριθμό επαναλήψεων.

neurons	lr	mc	Epochs	MSE (train)	MSE (test)	accurancy (train)	accurancy (test)
37	0.1	0.1	1000	0.0532	0.0563	0.6474	0.6178
74	0.1	0.1	1000	0.0469	0.0497	0.7043	0.6778

Από τα παραπάνω μπορούμε να συμπεράνουμε ότι με αρκετή εκπαίδευση του δικτύου -1000 επαναλήψεις αντί για 100- τα πλεονεκτήματα της χρήσης περισσοτέρων νευρώνων στο κρυφό επίπεδο γίνονται πια εμφανή. Σε ένα δύσκολο πρόβλημα όπως αυτό της αναγνώρισης γραφικού χαρακτήρα επιβάλλονται πολλαπλές επαναλήψεις στο στάδιο της εκπαίδευσης του δικτύου, ώστε να πάρουμε ικανοποιητικά αποτελέσματα. Για να μπορέσουμε να χρησιμοποιήσουμε για πρακτικούς σκοπούς το σχεδιασμένο νευρωνικό δίκτυο θα έπρεπε να εξετάσουμε και το ενδεχόμενο εκπαίδευσης του δικτύου σε ακόμη μεγαλύτερο αριθμό επαναλήψεων, λαμβάνοντας πάντοτε υπ' όψην και το ενδεχόμενο υπερειδίκευσης των νευρώνων του.

Οι κρυφοί νευρώνες δρουν σαν ανιχνευτές χαρακτηριστικών, παίζοντας κρίσιμο ρόλο στη λειτουργία ενός Perceptron πολλών επιπέδων. Καθώς προχωρά η διαδικασία μάθησης, οι κρυφοί νευρώνες εξάγουν προοδευτικά τα πιο σημαντικά χαρακτηριστικά από τα διανύσματα εισόδου. Αυτό το κάνουν εκτελώντας έναν μη γραμμικό μετασχηματισμό στα δεδομένα εισόδου, σ'ένα νέο χώρο που αποκαλείται «χώρος χαρακτηριστικών». Σάυτό το νέο χώρο, οι κλάσεις χαρακτηριστικών που ενδιαφέρουν μια εργασία ταξινόμησης προτύπων μπορούν να ξεχωριστούν από οτιδήποτε άλλο θα μπορούσε να υπάρχει στον αρχικό χώρο δεδομένων εισόδου.

3.3 Παράρτημα (Πλήρεις κώδικες ερωτημάτων)

Ερώτημα 1.1

```
% Read the ASCII-delimited numeric data file
% and return the data in output matrix.

Train = dlmread('optdigits.tra');
Test = dlmread('optdigits.tes');
```

Ερώτημα 1.2

```
% Read only input data from file and return the data in output matrix.
trainInput = dlmread('optdigits.tra', ',', [0,0,3822,63]);
testInput = dlmread('optdigits.tes', ',', [0,0,1796,63]);

% Process matrices by mapping row minimum and maximum values to [-1 1]
Normalized_trainInput = mapminmax(trainInput);
Normalized_testInput = mapminmax(testInput);
```

Ερώτημα 1.3

```
% Read only output data from file and return the data in output matrix
trainOutput = dlmread('optdigits.tra', ',', 0, 64);
testOutput = dlmread('optdigits.tes', ',', 0, 64);

% Create array of all zeros
transformed_trainOutput = zeros(3823,10);
transformed_testOutput = zeros(1797,10);

% Create train output matrix
for k = 1:numel(trainOutput)
    v = trainOutput(k);
    transformed_trainOutput(k,10 - v) = 1;
end

% Create test output matrix
for k = 1:numel(testOutput)
    v = testOutput(k);
    transformed_testOutput(k,10 - v) = 1;
end
```

Ερώτημα 1.4

```
%Concatenate arrays horizontally
newTrain = horzcat(Normalized_trainInput, transformed_trainOutput);
newTest = horzcat(Normalized_testInput, transformed_testOutput);
```

```
% sets the "seed" of the pseudo-random number generator to 7293.
% Once the seed is set to a given value, the algorithm always produces
% the same sequence of random numbers. This is useful if we need to use
% the same random numbers more than once, or to produce identical runs
% of the same simulation.
rand('seed', 7293);

% Random permutation synolou ekpaideysis
shuffled_newTrain = newTrain(randperm(size(newTrain,1)),:);

% Random permutation synolou elegxou
shuffled_newTest = newTest(randperm(size(newTest,1)),:);

% Create train matrix
train_matrix = shuffled_newTrain(:,1:64);

% Create target matrix
target_matrix = shuffled_newTrain(:,65:74);

% Create test matrix
test_matrix = shuffled_newTest(:,1:64);

% Create test target matrix
test_target_matrix = shuffled_newTest(:,65:74);

% Prepare data for training and simulation
% Trasposition of matrices is required for trainning
% and simulation functions to work correctly
train_matrix = train_matrix';
target_matrix = target_matrix';
test_matrix = test_matrix';
test_target_matrix = test_target_matrix';
```

Ερώτημα 3.1

```
% Create a feed-forward backpropagation network
net = newff(minmax(train_matrix),[37,10],{'logsig','logsig'},'traingdm');

net = init(net);

% We set the maximum training epoch
% the maximum number of failed validation checks
% and the minimum gradient
net.trainParam.epochs = 100;
net.trainParam.max_fail = 10;
net.trainParam.min_grad = 1e-10;

% learning rate = 0.001
% momentume     = 0.9
net.trainParam.lr = 0.01;
net.trainParam.mc = 0.9;
[results3a(1, 1), results3a(1, 2), results3a(1, 3), results3a(1, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;

% learning rate = 0.001
% momentume     = 0.6
net.trainParam.lr = 0.01;
net.trainParam.mc = 0.6;
[results3a(2, 1), results3a(2, 2), results3a(2, 3), results3a(2, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;

% learning rate = 0.001
% momentume     = 0.1
net.trainParam.lr = 0.01;
net.trainParam.mc = 0.1;
[results3a(3, 1), results3a(3, 2), results3a(3, 3), results3a(3, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
```

```
    ;

% learning rate = 0.01
% momentume     = 0.9
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.9;
[results3a(4, 1), results3a(4, 2), results3a(4, 3), results3a(4, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;

% learning rate = 0.01
% momentume     = 0.6
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.6;
[results3a(5, 1), results3a(5, 2), results3a(5, 3), results3a(5, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;

% learning rate = 0.01
% momentume     = 0.1
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.1;
[results3a(6, 1), results3a(6, 2), results3a(6, 3), results3a(6, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;
```

Ερώτημα 3.2

```
% Create a feed-forward backpropagation network
net = newff(minmax(train_matrix),[74,10],{'logsig','logsig'},'traingdm');

net = init(net);

% We set the maximum training epoch
% the maximum number of failed validation checks
% and the minimum gradient
net.trainParam.epochs = 100;
net.trainParam.max_fail = 10;
net.trainParam.min_grad = 1e-10;

% learning rate = 0.01
% momentume     = 0.1
net.trainParam.lr = 0.1;
net.trainParam.mc = 0.1;
[results3b(6, 1), results3b(6, 2), results3b(6, 3), results3b(6, 4)] = trainNTest(net,
    train_matrix, test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput)
    ;
```

Η ακόλουθη συνάρτηση χρησιμοποιείται επανειλημμένα στα ερωτήματα 3.1 και 3.2 για να εκπαιδεύσει το δίκτυο, να εκτελέσει τα πειράματα (5 φορές το καθένα) και να επιστρέψει τις ζητούμενες μετρικές.

```
function [ accurTrain, accurTest, perfTrain, perfTest ] = trainNTest( net, train_matrix,
    test_matrix, target_matrix, test_target_matrix, trainOutput, testOutput )

    % Define and initialize output variables
    accurTrain = 0;
    accurTest = 0;
    perfTrain = 0;
    perfTest = 0;

    % We repeat the test 5 times and return the mean
    % value of each resulting value
    for i=1:5
        % Train neural network
        [net,] = train(net, train_matrix, target_matrix);

        % Simulate neural network (train)
        yTrain = sim(net, train_matrix);
```

```
            % As a performance measure we use the Euclidean norm (train)
            perfTrain = perfTrain + mse(net, target_matrix, yTrain);

            % Initialize random number generator
            % 3879 + 3414 = 7293
            rand('seed', 7293);

            % Compute the accuracy of the trainning data set
            target_in_nums = trainOutput(randperm(size(trainOutput,1)),:);
            target_in_nums = target_in_nums';
            accurTrain = accurTrain + percent(yTrain, target_in_nums);

            % Simulate neural network (test)
            yTest = sim(net, test_matrix);

            % As a performance measure we use the Euclidean norm (test)
            perfTest = perfTest + mse(net, test_target_matrix, yTest);

            % Calculate test accuracy
            test_target_in_nums = testOutput(randperm(size(testOutput,1)),:);
            test_target_in_nums = test_target_in_nums';
            accurTest = accurTest + percent(yTest, test_target_in_nums);
        end

        % Divide by the number of repetitions
        perfTrain = perfTrain/5;
        perfTest = perfTest/5;
        accurTrain = accurTrain/5;
        accurTest = accurTest/5;
    end
```