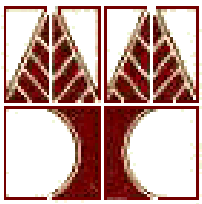
Ατομική Διπλωματική Εργασία

Πρόβλεψη αποτελεσμάτων αγώνων καλαθόσφαιρας, με τη χρήση τεχνητών νευρωνικών δικτύων

Ηλιανός Πελαβάς

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΪΟΣ 2014

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Πρόβλεψη αποτελεσμάτων αγώνων καλαθόσφαιρας, με τη χρήση τεχνητών νευρωνικών δικτύων

Ηλιανός Πελαβάς

Επιβλέπων Καθηγητής

Δρ. Χρίστος Χριστοδούλου

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

ΜΑΪΟΣ 2014

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Χρίστο Χριστοδούλου που με επέλεξε για την υλοποίηση του συγκεκριμένου θέματος και που καθ' όλη τη διάρκεια της εκπόνησης της Ατομικής Διπλωματικής μου Εργασίας, με στήριξε και με καθοδήγησε, ώστε να φτάσω στο επιθυμητό αποτέλεσμα.

Περίληψη

Αυτή η διπλωματική εργασία έχει σαν σκοπό, την υλοποίηση τεχνητών νευρωνικών δικτύων τα οποία θα είναι σε θέση να προβλέπουν επιτυχώς, σημεία στο στοίχημα που αφορά την καλαθόσφαιρα και συγκεκριμένα του Αμερικανικού πρωταθλήματος NBA (National Basketball Association). Το πρόγραμμα που θέλαμε να υλοποιήσουμε, θα έπρεπε να αυξάνει το αρχικό μας κεφάλαιο και να απέφερε κέρδος.

Κάτι τέτοιο δεν είχε υλοποιηθεί στο παρελθόν στο Πανεπιστήμιο Κύπρου, αλλά ούτε σε μεγάλο βαθμό στο εξωτερικό. Μετά από σχετική αναζήτηση που έγινε στο διαδίκτυο, βρέθηκαν κάποιες αναφορές για το πρόβλημα χωρίς όμως να ήταν διαθέσιμο κάπου υλοποιημένο. Υπήρχε όμως προηγούμενη δουλειά που αφορούσε το ποδοσφαιρικό στοίχημα. Αφού μελέτησα τη δουλειά δύο ατόμων, απόφοιτων του Πανεπιστημίου Κύπρου, αποφάσισα το είδος στοιχήματος με το οποίο ασχολήθηκα και αυτό ήταν το OVER/UNDER του συνολικού αριθμού που σκοράρουν οι δύο αγωνιζόμενες ομάδες σε ένα αγώνα. Ακόμη επέλεξα και το είδος των νευρωνικών δικτύων που χρησιμοποίησα.

Υλοποίησα τον αλγόριθμο Back-Propagation και χρησιμοποίησα τα Radial Basis Function Networks (RBF Networks) για την επίλυση του προβλήματος κάτι που διδάχτηκα στο μάθημα Υπολογιστικά Συστήματα Μάθησης. Ήταν μέθοδοι που χρησιμοποιήθηκαν στο παρελθόν για το πρόβλημα πρόβλεψης αποτελεσμάτων για ποδοσφαιρικούς αγώνες. Για την υλοποίηση των δύο αλγόριθμων και τη δημιουργία των αρχείων δεδομένων που χρησιμοποιούσαν τα δύο δίκτυα, χρησιμοποίησα τη γλώσσα προγραμματισμού Java.

Για κάθε δίκτυο, πήραμε πολλά αποτελέσματα, ανάλογα με τις παραμέτρους που δίναμε κάθε φορά. Τα επεξεργαστήκαμε, τα αναλύσαμε και καταλήξαμε στα καλύτερα δυνατά για κάθε αλγόριθμο. Προσπαθήσαμε να δούμε ποιες παράμετροι επηρεάζουν τον κάθε αλγόριθμο και να τις προσαρμόσουμε ανάλογα, ώστε να αυξήσουμε τα ποσοστά επιτυχίας. Λόγω του μεγάλου αριθμού αποτελεσμάτων, συμπεριέλαβα σε αυτό το έγγραφο, αυτά που θεωρούσα τα πιο σημαντικά.

Τα αποτελέσματα που πήραμε από τον αλγόριθμο Back-Propagation, έφτασαν το ποσοστό επιτυχίας 70%. Αυτό το ποσοστό κρίνεται πολύ επιτυχημένο, αφού με αποδόσεις 1.90 που είναι αυτές που προσφέρονται, το καθαρό κέρδος φτάνει το 33%.

Τα αποτελέσματα από τον αλγόριθμο RBF δικτύων, δεν ήταν τα αναμενόμενα, αφού πήραμε ποσοστό επιτυχίας 55%. Κάτι που μας επιστρέφει πολύ μικρό κέρδος, 4,5% δηλαδή. Με τέτοιο ποσοστό επιτυχίας δεν αξίζει να ρισκάρουμε να επενδύσουμε τα χρήματά μας.

Και στις δύο περιπτώσεις τα αποτελέσματα πάρθηκαν με τη χρήση 10 εισόδων και 20 κρυφών νευρώνων. Θα εξηγήσουμε στη συνέχεια πως έγινε η υλοποίηση και με βάση ποιες άλλες παραμέτρους προέκυψαν αυτά τα αποτελέσματα.

Για τον αλγόριθμο Back-Propagation κρίνω ότι δεν υπάρχουν πολλά περιθώρια βελτίωσης όμως για τον αλγόριθμο RBF δικτύων, ίσως σε μεταγενέστερο στάδιο αυτά τα αποτελέσματα να βελτιωθούν.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο 1

Εισαγωγή

1.1 Γενική Εισαγωγή

Όπως προαναφέρθηκε, το θέμα της διπλωματικής αυτής εργασίας, είναι η πρόβλεψη αποτελεσμάτων αγώνων NBA (National Basketball Association), με τη χρήση νευρωνικών δικτύων. Νευρωνικό δίκτυο ονομάζεται ένα κύκλωμα διασυνδεδεμένων νευρώνων. Βιολογικά, πρόκειται για ένα τμήμα νευρικού ιστού. Στην περίπτωση τεχνητών νευρώνων, πρόκειται για ένα αφηρημένο αλγοριθμικό κατασκεύασμα το οποίο εμπίπτει στον τομέα της υπολογιστικής νοημοσύνης. Στόχος του νευρωνικού δικτύου είναι η επίλυση κάποιου υπολογιστικού προβλήματος, ή της υπολογιστικής Νευροεπιστήμης, όταν στόχος είναι η υπολογιστική προσομοίωση της λειτουργίας των βιολογικών νευρωνικών δικτύων με βάση κάποιο μαθηματικό μοντέλο τους [11].

Το νευρωνικό δίκτυο, είναι ένα δίκτυο που αποτελείται από απλούς υπολογιστικούς κόμβους (νευρώνες, νευρώνια), διασυνδεδεμένους μεταξύ τους. Οι νευρώνες, είναι τα δομικά στοιχεία του δικτύου. Κάθε κόμβος δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές (είτε από άλλους νευρώνες, είτε από το περιβάλλον), επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο.

Το κύριο χαρακτηριστικό των νευρωνικών δικτύων είναι η εγγενής ικανότητα μάθησης. Ως μάθηση μπορεί να οριστεί η σταδιακή βελτίωση της ικανότητας του δικτύου να επιλύει κάποιο πρόβλημα. Η μάθηση επιτυγχάνεται μέσω της εκπαίδευσης, μίας επαναληπτικής διαδικασίας σταδιακής προσαρμογής των

παραμέτρων του δικτύου σε τιμές κατάλληλες ώστε να επιλύεται με επαρκή επιτυχία το προς εξέταση πρόβλημα. Αφού ένα δίκτυο εκπαιδευτεί, οι παράμετροί του συνήθως «παγώνουν» στις κατάλληλες τιμές και από εκεί κι έπειτα είναι σε λειτουργική κατάσταση. Το ζητούμενο είναι το λειτουργικό δίκτυο να χαρακτηρίζεται από μία ικανότητα δηλαδή, μπορεί να δώσει ορθές εξόδους, για εισόδους διαφορετικές από αυτές που εκπαιδεύτηκε.

Τα νευρωνικά δίκτυα, εφαρμόζονται σε πολλούς επιστημονικούς τομείς, όπως για παράδειγμα στην Ιατρική όπου χρησιμοποιούνται για διαγνώσεις, στα Οικονομικά, για χρηματιστηριακές προβλέψεις , στην Πληροφορική, για αναγνώριση προτύπων, στη Βιολογία για μελέτη νευρώνων.

Η εκπαίδευση των Νευρωνικών Δικτύων, μπορεί να γίνει με τρεις τρόπους.

α) Επιβλεπόμενη μάθηση:

Συνδυάζει έναν εξωτερικό εκπαιδευτή και τη συνολική ή γενικευμένη πληροφορία. Προσαρμόζουμε τα βάρη ώστε να ελαχιστοποιήσουμε τη διαφορά μεταξύ επιθυμητών και πραγματικών αποτελεσμάτων για κάθε μοτίβου εισόδου.

β) Μη Επιβλεπόμενη μάθηση:

Το δίκτυο είναι σε θέση να εξερευνήσει στατιστικές κανονικότητες και αυτόματα αναπτύσσει διάφορους τρόπους για να αναπαραστήσει κλάσεις εισόδων.

γ) Ενισχυτική μάθηση:

Το δίκτυο λαμβάνει ένα σήμα επιβράβευσης ή τιμωρίας. Τα βάρη αλλάζουν ώστε να αναπτυχθεί μια input/output συμπεριφορά ώστε να αυξηθεί η πιθανότητα για λήψη επιβράβευσης και να ελαχιστοποιηθεί η πιθανότητα για λήψη τιμωρίας.

1.2 Προηγούμενη Δουλειά

Για την εκπόνηση της Διπλωματικής Εργασίας, έγινε μελέτη κατά πόσο το πρόβλημα είναι εφικτό να λυθεί ή αν είχε λυθεί στο παρελθόν. Όσον αφορά το πρώτο ερώτημα, κατόπιν συζήτησης με τον καθηγητή μου Δρ. Χρίστο Χριστοδούλου, κατέληξα στο συμπέρασμα ότι υπάρχει λύση. Το θέμα ήταν πόσο ικανοποιητική θα ήταν αυτή και αν μπορούσαμε να καταλήξουμε σε κέρδος.

Όσον αφορά την προηγούμενη δουλειά, δεν βασίστηκα σε διπλωματική εργασία που αφορούσε πρόβλεψη για αποτελέσματα αγώνων NBA. Παρόλα αυτά, μελέτησα τη δουλειά δύο φοιτητών του Πανεπιστημίου Κύπρου, οι οποίοι είχαν ασχοληθεί στο παρελθόν με πρόβλεψη αποτελεσμάτων ποδοσφαιρικών αγώνων. Η δουλειά τους ήταν στα πλαίσια της ατομικής διπλωματικής εργασίας τους.

Αρχικά μελέτησα τη δουλειά του Θεοφάνη Νεοκλέους [1], ο οποίος το 2007, ασχολήθηκε με την πρόβλεψη του συνολικού αριθμού τερμάτων που σημειώνονταν σε κάθε ποδοσφαιρικό αγώνα. Τα ποσοστά επιτυχίας του σε σχέση με τις προηγούμενες εργασίες που μελέτησε, ήταν αυξημένα, αφού κατάφερε να αποφύγει λάθη που έγιναν στο παρελθόν και να προσθέσει στοιχεία που θα του απέφεραν ένα πιο επιτυχές αποτέλεσμα. Ο Θεοφάνης Νεοκλέους [1], χρησιμοποίησε κυρίως τον αλγόριθμο Back Propagation και εκπαίδευσε τα δίκτυα του με βάση τους αγώνες της Championship (κατηγορία πρωταθλήματος στην Αγγλία).

Ακολούθως, μελέτησα τη δουλειά του Ανδρέα Ιακώβου [2] ο οποίος το 2010 αφού έλαβε υπόψη τις εργασίες φοιτητών, ακολούθησε κατά κύριο λόγο την πορεία του Θεοφάνη Νεοκλέους [1], με σκοπό να λύσει το ίδιο πρόβλημα, χρησιμοποιώντας αλγόριθμο RBF δικτύων. Κατάφερε να βελτιώσει τα αποτελέσματα ακόμα περισσότερο και να πετύχει σημαντικό κέρδος μέσω των προβλέψεων των δικτύων του.

Έχοντας σαν δεδομένα τα όσα έλαβαν υπόψη οι δύο αυτοί φοιτητές, τα όσα πέτυχαν και τα όσα δεν πέτυχαν, θα προσπαθήσουμε να υλοποιήσουμε τους αλγόριθμους αυτούς, για διαφορετικό πρόβλημα και διαφορετικές εισόδους με την ελπίδα να είναι το ίδιο ή ακόμη και περισσότερο κερδοφόροι στην περίπτωση της καλαθόσφαιρας.

1.3 Στόχος Διπλωματικής Εργασίας

Στόχος της Διπλωματικής Εργασίας είναι η κατασκευή ενός κερδοφόρου συστήματος το οποίο θα βασίζεται στην επιτυχή πρόβλεψη αποτελεσμάτων των αγώνων NBA με τη χρήση νευρωνικών δικτύων. Ασφαλώς κάτι τέτοιο δεν είναι εύκολο αφού πολλές φορές, στην έκβαση ενός αποτελέσματος, παίζει σημαντικό ρόλο η τύχη ή κάποιοι αστάθμητοι παράγοντες όπως ο τραυματισμός κάποιου παίκτη, η απουσία του κόσμου, η έλλειψη κινήτρου. Εντούτοις, τα νευρωνικά δίκτυα, μπορεί να δώσουν κάποιες επιτυχείς προβλέψεις που μπορούν να αυξήσουν το κεφάλαιό μας.

Μετά από έρευνα που έγινε, στο διαδίκτυο δεν υπάρχει κάποια ιστοσελίδα που να δίνει προγνωστικά για αγώνες NBA τα οποία βασίζονται στη χρήση νευρωνικών δικτύων. Αντίθετα, όσο αφορά το ποδόσφαιρο υπάρχουν συγκεκριμένες ιστοσελίδες που το παρέχουν. Υπάρχουν κάποιες αναφορές σε διπλωματικές εργασίες που αφορούν το NBA, όμως δεν είναι προσβάσιμες.

Έτσι, κρίνω σημαντική την Διπλωματική αυτή Εργασία, αφού είναι κάτι καινούργιο στο χώρο των νευρωνικών δικτύων και η απόδειξη ότι αυτά μπορούν να προβλέψουν τέτοια αποτελέσματα επιτυχώς, θα είναι πολύ χρήσιμη.

Ακόμη, η αύξηση του κεφαλαίου μας και το καθαρό κέρδος, δεν μπορούν να μας αφήσουν αδιάφορους, αφού αν επιτευχθούν τα επιθυμητά ποσοστά επιτυχίας, τότε η χρησιμοποίηση του νευρωνικού αυτού δικτύου, θα είναι καλύτερη από κάθε είδους επένδυση σε τράπεζα που προσφέρει τόκο, ή σε ακίνητα που μπορεί να χάσουν την αξία τους.

1.4 Το Πρόβλημα

Το πρωτάθλημα του NBA, οργανώνεται με τον ακόλουθο τρόπο.

Αποτελείται από την κανονική περίοδο και τα playoffs. Εμείς θα ασχοληθούμε μόνο με την κανονική περίοδο.

Υπάρχουν τριάντα ομάδες οι οποίες χωρίζονται ισάριθμα σε δύο συνομοσπονδίες. Κάθε συνομοσπονδία αποτελείται από τρεις κατηγορίες δηλαδή έχουμε σύνολο έξι κατηγορίες με πέντε ομάδες η κάθε μια.

Κάθε ομάδα αναμετρείται τέσσερις φορές με τις ομάδες της κατηγορίας της, δύο φορές με όλες τις ομάδες την άλλης συνομοσπονδίας και με βάση προκλήρωσης που γίνεται κάθε πέντε χρόνια αντιμετωπίζει τέσσερις φορές τις έξι από τις υπόλοιπες δέκα ομάδες της συνομοσπονδίας της και τρεις φορές τις υπόλοιπες τέσσερις.

Συνολικά κάθε ομάδα δίνει 82 αγώνες στην κανονική περίοδο, δηλαδή ο συνολικός αριθμός των αγώνων είναι 1230. Το ποιες ομάδες πάνε στα playoffs και τι γίνεται εκεί, δεν μας απασχολεί.

Όπως έχουμε πει, ο στόχος είναι η επίτευξη κέρδους. Υπάρχει πληθώρα πονταρισμάτων όπως στη νίκη της μιας ή της άλλης ομάδας με διαφορά πόντων, under/over ενός συνολικού αριθμού πόντων, στοίχημα για τον νικητή της κάθε περιόδου, του ημιχρόνου, την περίοδο με το μεγαλύτερο σκορ, το ποια ομάδα θα φτάσει πρώτη στους 20 πόντους, στο συνδυασμό ημιχρόνου/τελικού νικητή και άλλα που δεν θα μας απασχολήσουν.

2013-2014 CONFERENCE REGULAR SEASON STANDINGS CREATED: SUNDAY APRIL 20, 10:06 AM

EASTERN CONFERENCE

Eastern	W	L	PCT	GB	CONF	DIV	HOME	ROAD	L 10	STREAK
Indiana 1e	56	26	0.683	0.0	38-14	12-4	35-6	21-20	4-6	W 2
Miami 2se	54	28	0.659	2.0	34-18	12-4	32-9	22-19	4-6	L 3
Toronto 3a	48	34	0.585	8.0	32-20	11-5	26-15	22-19	7-3	L 1
Chicago 4x	48	34	0.585	8.0	35-17	11-5	27-14	21-20	8-2	L 1
Washington 5x	44	38	0.537	12.0	33-19	10-6	22-19	22-19	7-3	W 4
Brooklyn 6x	44	38	0.537	12.0	26-26	9-7	28-13	16-25	5-5	L 2
Charlotte 7x	43	39	0.524	13.0	30-22	6-10	25-16	18-23	8-2	W 3
Atlanta 8x	38	44	0.463	18.0	28-24	8-8	24-17	14-27	7-3	W 1
New York o	37	45	0.451	19.0	26-26	10-6	19-22	18-23	7-3	W 4
Cleveland o	33	49	0.402	23.0	21-31	7-9	19-22	14-27	5-5	W 1
Detroit o	29	53	0.354	27.0	23-29	6-10	17-24	12-29	3-7	L 4
Boston o	25	57	0.305	31.0	21-31	5-11	16-25	9-32	2-8	L 2
Orlando o	23	59	0.280	33.0	17-35	4-12	19-22	4-37	3-7	L 4
Philadelphia o	19	63	0.232	37.0	14-38	5-11	10-31	9-32	4-6	W 2
Milwaukee o	15	67	0.183	41.0	12-40	4-12	10-31	5-36	1-9	L 3

WESTERN CONFERENCE

Western	W	L	PCT	GB	CONF	DIV	HOME	ROAD	L 10	STREAK
San Antonio 1w	62	20	0.756	0.0	38-14	12-4	32-9	30-11	6-4	L 2
Oklahoma City 2nw	59	23	0.720	3.0	36-16	11-5	34-7	25-16	6-4	W 1
L.A. Clippers 3p	57	25	0.695	5.0	36-16	12-4	34-7	23-18	7-3	L 1
Houston 4x	54	28	0.659	8.0	31-21	11-5	33-8	21-20	5-5	L 1
Portland 5x	54	28	0.659	8.0	31-21	13-3	31-10	23-18	9-1	W 5
Golden State 6x	51	31	0.622	11.0	31-21	11-5	27-14	24-17	6-4	W 2
Memphis 7x	50	32	0.610	12.0	29-23	4-12	27-14	23-18	7-3	W 5
Dallas 8x	49	33	0.598	13.0	29-23	9-7	26-15	23-18	6-4	L 1
Phoenix o	48	34	0.585	14.0	28-24	8-8	26-15	22-19	5-5	W 1
Minnesota o	40	42	0.488	22.0	23-29	7-9	24-17	16-25	4-6	L 3
Denver o	36	46	0.439	26.0	20-32	5-11	22-19	14-27	4-6	L 2
New Orleans o	34	48	0.415	28.0	15-37	4-12	22-19	12-29	2-8	W 2
Sacramento o	28	54	0.341	34.0	15-37	3-13	17-24	11-30	3-7	L 1
L.A. Lakers o	27	55	0.329	35.0	15-37	6-10	14-27	13-28	3-7	W 2
Utah o	25	57	0.305	37.0	13-39	4-12	16-25	9-32	2-8	W 1

Σχήμα 1.1 – Οι δύο συνομοσπονδίες
του πρωταθλήματος

Στην πιο πάνω εικόνα, βλέπουμε τις ομάδες του πρωταθλήματος 2013-2014 χωρισμένες στις δύο συνομοσπονδίες, την ανατολική και τη δυτική. Παρατηρούμε ότι η κάθε συνομοσπονδία έχει δεκαπέντε ομάδες.

2013-2014 DIVISION REGULAR SEASON STANDINGS CREATED: SUNDAY APRIL 20, 10:06 AM

EASTERN CONFERENCE										
Atlantic	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
Toronto 3a	48	34	0.585	0.0	32-20	11-5	26-15	22-19	7-3	L 1
Brooklyn 6x	44	38	0.537	4.0	26-26	9-7	28-13	16-25	5-5	L 2
New York °	37	45	0.451	11.0	26-26	10-6	19-22	18-23	7-3	W 4
Boston °	25	57	0.305	23.0	21-31	5-11	16-25	9-32	2-8	L 2
Philadelphia °	19	63	0.232	29.0	14-38	5-11	10-31	9-32	4-6	W 2
Central	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
Indiana 1e	56	26	0.683	0.0	38-14	12-4	35-6	21-20	4-6	W 2
Chicago 4x	48	34	0.585	8.0	35-17	11-5	27-14	21-20	8-2	L 1
Cleveland °	33	49	0.402	23.0	21-31	7-9	19-22	14-27	5-5	W 1
Detroit °	29	53	0.354	27.0	23-29	6-10	17-24	12-29	3-7	L 4
Milwaukee °	15	67	0.183	41.0	12-40	4-12	10-31	5-36	1-9	L 3
Southeast	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
Miami 2se	54	28	0.659	0.0	34-18	12-4	32-9	22-19	4-6	L 3
Washington 5x	44	38	0.537	10.0	33-19	10-6	22-19	22-19	7-3	W 4
Charlotte 7x	43	39	0.524	11.0	30-22	6-10	25-16	18-23	8-2	W 3
Atlanta 8x	38	44	0.463	16.0	28-24	8-8	24-17	14-27	7-3	W 1
Orlando °	23	59	0.280	31.0	17-35	4-12	19-22	4-37	3-7	L 4
WESTERN CONFERENCE										
Northwest	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
Oklahoma City 2nw	59	23	0.720	0.0	36-16	11-5	34-7	25-16	6-4	W 1
Portland 5x	54	28	0.659	5.0	31-21	13-3	31-10	23-18	9-1	W 5
Minnesota °	40	42	0.488	19.0	23-29	7-9	24-17	16-25	4-6	L 3
Denver °	36	46	0.439	23.0	20-32	5-11	22-19	14-27	4-6	L 2
Utah °	25	57	0.305	34.0	13-39	4-12	16-25	9-32	2-8	W 1
Pacific	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
L.A. Clippers 3p	57	25	0.695	0.0	36-16	12-4	34-7	23-18	7-3	L 1
Golden State 6x	51	31	0.622	6.0	31-21	11-5	27-14	24-17	6-4	W 2
Phoenix °	48	34	0.585	9.0	28-24	8-8	26-15	22-19	5-5	W 1
Sacramento °	28	54	0.341	29.0	15-37	3-13	17-24	11-30	3-7	L 1
L.A. Lakers °	27	55	0.329	30.0	15-37	6-10	14-27	13-28	3-7	W 2
Southwest	**W**	**L**	**PCT**	**GB**	**CONF**	**DIV**	**HOME**	**ROAD**	**L 10**	**STREAK**
San Antonio 1w	62	20	0.756	0.0	38-14	12-4	32-9	30-11	6-4	L 2
Houston 4x	54	28	0.659	8.0	31-21	11-5	33-8	21-20	5-5	L 1
Memphis 7x	50	32	0.610	12.0	29-23	4-12	27-14	23-18	7-3	W 5
Dallas 8x	49	33	0.598	13.0	29-23	9-7	26-15	23-18	6-4	L 1
New Orleans °	34	48	0.415	28.0	15-37	4-12	22-19	12-29	2-8	W 2

Σχήμα 1.2 – Οι 6 κατηγορίες του πρωταθλήματος

Σε αυτή την εικόνα, παρατηρούμε ότι κάθε συνομοσπονδία, περιέχει τρεις κατηγορίες, άρα και κάθε κατηγορία πέντε ομάδες.

Οι αγώνες κάθε ομάδας, ορίζονται με βάση αυτά που αναφέραμε πιο πάνω.

Δεν προκαθορίζουμε το είδος στοιχήματος με το οποίο θα ασχοληθούμε αφού θα δοκιμάσουμε διάφορα και ανάλογα με την επιτυχία που έχουμε για το κάθε ένα, θα προχωρήσουμε.

Για να παραχθεί κάποιο αποτέλεσμα από το δίκτυο μας, θα πρέπει να δώσουμε σε αυτό, σαν είσοδο κάποια δεδομένα. Δεδομένα θα αποτελέσουν τα αποτελέσματα των αγώνων NBA τα τελευταία 10 χρόνια τα οποία έχουν ήδη αντληθεί από συγκεκριμένη ιστοσελίδα στο διαδίκτυο (www.basketball-reference.com) [10].

Θα τύχουν επεξεργασίας για να μπορούν να δοθούν οι απαιτούμενες παράμετροι στο δίκτυο.

Μερικά παραδείγματα επεξεργασμένων δεδομένων που σκοπεύουμε να χρησιμοποιήσουμε είναι:
Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της, σε όλους τους προηγούμενους αγώνες της χρονιάς.
Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας, σε όλους τους προηγούμενους αγώνες της χρονιάς.
Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της, σε όλους τους προηγούμενους αγώνες της χρονιάς.
Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας, σε όλους τους προηγούμενους αγώνες της χρονιάς.
Ποσοστό παιχνιδιών της γηπεδούχου ομάδας στην έδρα της στα οποία σημειώθηκε πάνω από ένας συγκεκριμένος αριθμός πόντων.

1.4 Επιτυχία

Τι μπορεί να αποτελεί επιτυχία στο πρόβλημά μας; Ασφαλώς η επίτευξη κέρδους. Η επίτευξη κέρδους έχει άμεση σχέση με τα ποσοστά επιτυχίας του δικτύου μας κάθως επίσης και με τις αποδόσεις των σημείων στα οποία ποντάρουμε. Δηλαδή, το δίκτυο μας μπορεί να προβλέπει ορθά 70% των αγώνων και να μην μας αποφέρει κέρδος ή να προβλέπει ορθά 50% και να αποφέρει. Σημασία έχει η

απόδοση του σημείου το οποίο προβλέπουμε. Ας δούμε πιο κάτω ένα πίνακα με μερικά παραδείγματα.

Ποντάρισμα	Απόδοση	Επιτυχία	Συνολικά Ποντάρίσματα	Επιστροφές	Κέρδος/Ζημιά
1	1,30	70%	100	91	-9
1	1,50	60%	100	90	-10
1	2	50%	100	100	0
1	1,90	65%	100	123,5	23,5
1	2,30	55%	100	126,5	26,5

Σχήμα 1.3- Παραδείγματα πονταρισμάτων

Όπως παρατηρούμε στον πίνακα, δίκτυο το οποίο είχε 70% ορθές προβλέψεις, δεν σημαίνει απαραίτητα και κερδοφόρο αφού προέβλεπε σωστά σημεία τα οποία ήταν χαμηλά σε απόδοση. Αντίθετα το δίκτυο με 55% ορθές προβλέψεις σε σημεία απόδοσης 2,30 στο ένα, μας απέφερε κέρδος τάξεως 26,5%. Συμπεραίνουμε λοιπόν ότι η επιτυχία έχει άμεσε σχέση με την απόδοση του σημείου που προβλέπουμε.

1.5 Παραδείγματα στοιχημάτων


NBA Sunday 20/04 Lines
Game Lines

			Spread	Totals	Money Line	
20 Apr 19:05	709	DAL Mavericks	+9.5 1.90	O 202.5 1.90	5.25	13
	710	SA Spurs	-9.5 1.90	U 202.5 1.90	1.18	
20 Apr 21:30	711	CHA Bobcats	+10.0 1.90	O 186.0 1.90	5.75	13
	712	MIA Heat	-10.0 1.90	U 186.0 1.90	1.15	
21 Apr 01:05	713	WAS Wizards	+4.5 1.90	O 179.5 1.90	2.67	13
	714	CHI Bulls	-4.5 1.90	U 179.5 1.90	1.52	
21 Apr 03:35	715	POR Trail Blazers	+5.5 1.90	O 215.0 1.90	2.95	13
	716	HOU Rockets	-5.5 1.90	U 215.0 1.90	1.42	

Σχήμα 1.4 – Παραδείγματα στοιχημάτων από εταιρεία

Σε αυτή την εικόνα, που είναι παρμένη από εταιρεία στοιχημάτων στο διαδίκτυο, παρουσιάζονται τέσσερις αγώνες NBA[12]. Παρατηρούμε τους αριθμούς με άσπρα γράμματα να αλλάζουν ανάλογα με τον αγώνα, αφού η εταιρία προβλέπει διαφορετική διακύμανση του σκορ. Με κίτρινα γράμματα, αναγράφεται η απόδοση για κάθε σημείο.

Ας ασχοληθούμε μόνο με τον πρώτο αγώνα στον οποίο αγωνίζεται η ομάδα San Antonio Spurs με την ομάδα Dallas Mavericks.

Σε αυτή την περίπτωση, φαβορί για να νικήσει είναι η ομάδα San Antonio Spurs, γι' αυτό και στην τρίτη στήλη (Money Line) δίνεται η απόδοση 1.18 για νίκη της, ενώ για την νίκη των Dallas Mavericks, η απόδοση είναι 5.25.

Στην πρώτη στήλη (Spread), για να εξισωθεί κατά κάποιο τρόπο η διαφορά δυναμικότητας των δύο ομάδων, δίνεται η επιλογή να ποντάρουμε σε νίκη των San Antonio Spurs αφαιρώντας τους 9.5 πόντους από αυτούς που θα πετύχουν παίρνοντας απόδοση 1.90 αντί 1.18. Έτσι αν οι San Antonio Spurs κερδίσουν με δέκα ή περισσότερους πόντους διαφορά θα κερδίσουμε και εμείς, ενώ αν κερδίσουν με λιγότερους από δέκα, ή χάσουν, χάνουμε. Επίσης μπορούμε να ποντάρουμε σε νίκη των Dallas Mavericks δίνοντας τους πλεονέκτημα 9.5 πόντων και να πάρουμε

απόδοση 1.90 αντί πέντε. Δηλαδή αν οι Dallas Mavericks χάσουν στην πραγματικότητα με λιγότερους από δέκα πόντους διαφορά, εμείς πάλι θα κερδίσουμε.

Η μεσαία στήλη (Totals), αφορά τους συνολικούς πόντους που θα επιτευχθούν στο παιχνίδι, χωρίς να υπολογίζεται παράταση σε περίπτωση ισοπαλίας.

Μπορούμε εύκολα να παρατηρήσουμε, ότι ανάλογα με τον αγώνα, βλέπουμε διαφορετικές αποδόσεις για κάθε σημείο, τόσο σε Money Line και Spread, όσο και στο Totals. Αυτό έχει να κάνει αποκλειστικά με την εικόνα των ομάδων, τη δυναμικότητα, την τάση που έχουν να σκοράρουν, τις απουσίες παικτών και διάφορους άλλους παράγοντες.

Εμείς θα ασχοληθούμε με τη στήλη Totals, την πρόβλεψη δηλαδή του συνολικού αριθμού πόντων που θα επιτευχθούν στον αγώνα. Ανάλογα με την τιμή πόντων που δίνεται από την εταιρεία, θα τρέχουμε το πρόγραμμα πολλές φορές, αλλάζοντας το κατώφλι, ελέγχοντας έτσι αν το συγκεκριμένο ζευγάρι που θέλουμε, μας δώσει τιμή μεγαλύτερη ή μικρότερη.

Κεφάλαιο 2

Νευρωνικά Δίκτυα

2.1 Ιστορική Αναδρομή

Η ιστορία των νευρωνικών δικτύων, ξεκινά από το 1943 όταν ο νευροφυσιολόγος Warren McCulloch και ο μαθηματικός Walter Pitts [3][7] έγραψαν ένα βιβλίο, για το πώς οι νευρώνες θα μπορούσαν να λειτουργήσουν . Για να περιγράψουν τη λειτουργία των νευρώνων στον εγκέφαλο, μοντελοποίησαν ένα απλό νευρωνικό δίκτυο με τη χρήση ηλεκτρικών κυκλωμάτων.

Το 1949 ο Donald Hebb [4], παρουσίασε το βιβλίο του «The Organization of Behavior», ένα έργο που επεσήμανε το γεγονός ότι οι νευρικές οδοί ενισχύονται κάθε φορά που χρησιμοποιούνται , μια έννοια εντελώς απαραίτητη για τους τρόπους με τους οποίους οι άνθρωποι μαθαίνουν. Υποστήριξε ότι αν δύο νευρώνες «πυροβολούν» ταυτόχρονα, η σύνδεση μεταξύ τους είναι ενισχυμένη.

Το 1950 κι αφού οι υπολογιστές αναπτύχθηκαν περισσότερο, κατέστη δυνατό, να προσομοιωθεί ένα υποθετικό νευρωνικό δίκτυο. Το πρώτο βήμα προς αυτή την κατεύθυνση, έγινε από τον Nathanial Rochester, στα ερευνητικά εργαστήρια της IBM. Η πρώτη προσπάθεια ήταν αποτυχημένη.

Το 1962, ο Widrow και ο Hoff [5], ανάπτυξαν μια διαδικασία μάθησης η οποία εξετάζει την τιμή πριν την ρυθμίσει το βάρος με τον κανόνα: Weight Change = (Pre-Weight line value) * (Error / (Number of Inputs)).

Το 1972 και 1982 Kohonen και Anderson ανάπτυξαν ένα δίκτυο, βασισμένο στην ιδέα ότι οι νευρώνες οργανώνονται από μόνοι τους για να ρυθμίσουν διάφορα και συγκεκριμένα μοτίβα.

Το πρώτο πολυεπίπεδο δίκτυο αναπτύχθηκε το 1975, με μη επιβλεπόμενη μάθηση.

Το 1982, ο John Hopfield του Caltech παρουσίασε ένα έγγραφο με την Εθνική Ακαδημία Επιστημών. Η προσέγγισή του ήταν να δημιουργήσει πιο χρήσιμες μηχανές, με τη χρήση αμφίδρομων γραμμών . Προηγουμένως , οι συνδέσεις μεταξύ των νευρώνων γίνονταν μόνο με ένα δρόμο.

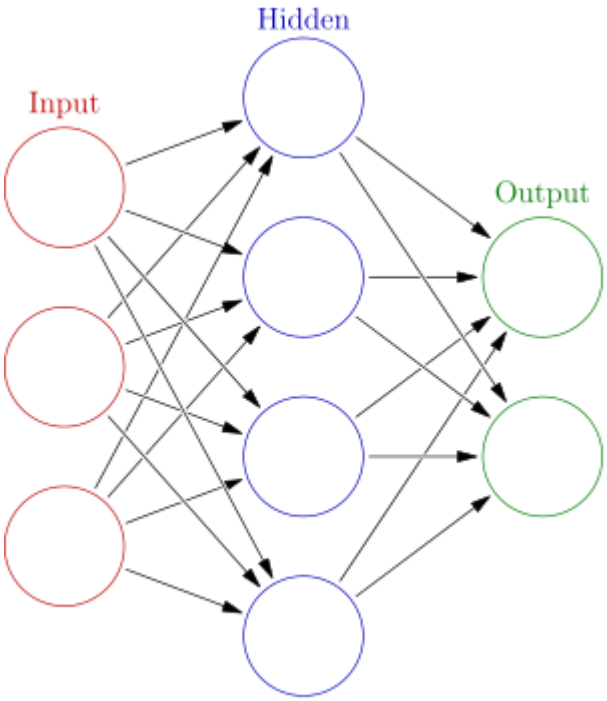
Το 1986 ο Rumelhart, ανέφερε την ιδέα για ανάπτυξη του αλγορίθμου με ανάστροφη διάδοση (back-propagation algorithm).

Το 1988 οι Broomhead και Lowe περιέγραψαν μία διαδικασία για το Σχεδιασμό FeedForward δικτύων (προς τα εμπρός τροφοδότηση) χρησιμοποιώντας *συναρτήσεις ακτινικής βάσης(RBF),* που είναι μια εναλλαγή των πολυεπίπεδων αισθητηρίων.

2.2 Τρόπος Λειτουργίας των Νευρωνικών Δικτύων

Τα Νευρωνικά Δίκτυα, ουσιαστικά μοντελοποιούν τον ανθρώπινο εγκέφαλο. Όπως δηλαδή λειτουργούν οι νευρώνες του εγκεφάλου, έτσι και εμείς προσπαθούμε να δημιουργήσουμε τεχνητούς νευρώνες, να τους συνδέσουμε και αφού καθορίσουμε ένα μαθηματικό ή υπολογιστικό μοντέλο, να τους κάνουμε να δέχονται συγκεκριμένες εισόδους, να τις επεξεργάζονται και να υπολογίζουν την έξοδο.

Η διαφορά στα Νευρωνικά Δίκτυα από υπόλοιπα προγράμματα είναι ότι δεν εκτελούμε μια διαδικασία με σκοπό να πάρουμε το μοναδικό και σωστό αποτέλεσμα. Τα Νευρωνικά Δίκτυα, μαθαίνουν μέσω παραδειγμάτων, όπως δηλαδή και ο ανθρώπινος εγκέφαλος, εκπαιδεύονται και προσπαθούν να προβλέπουν την έξοδο, βάση λογικής.



Σχήμα 2.1 – Νευρωνικό δίκτυο με συνδεδεμένους κόμβους

Πιο πάνω βλέπουμε ένα παράδειγμα νευρωνικού δικτύου με συνδεδεμένου κόμβους σε επίπεδο εισόδου, κρυφό επίπεδο και επίπεδο εξόδου. Οι κόμβοι (νευρώνες), λειτουργούν όπως και οι νευρώνες του ανθρώπινου εγκεφάλου.

Τα Νευρωνικά Δίκτυα, έχουν τη δυνατότητα να γενικεύουν μέσα από τα παραδείγματα και να δίνουν απαντήσεις με καλύτερη προσέγγιση σε διάφορα προβλήματα. Ακόμη, είναι ανθεκτικά σε σφάλματα αφού αν υπάρξει απώλεια ενός νευρώνα δεν σημαίνει απαραίτητα ότι θα σταματήσει η εκτέλεση του δικτύου.

2.3 Κατηγορίες Μάθησης Νευρωνικών Δικτύων

Σε αυτό το κομμάτι, θα περιγράψουμε αναλυτικότερα τις κατηγορίες μάθησης Νευρωνικών Δικτύων που αναφέραμε πιο πάνω.

Επιβλεπόμενη μάθηση:

Η επιβλεπόμενη μάθηση(supervised learning), είναι η μέθοδος κατά την οποία το σύστημα, μαθαίνει με βάση κάποια δεδομένα εισόδου στη φάση εκπαίδευσης, τα οποία αποτελούν παράδειγμα. Κάθε παράδειγμα, περιέχει ένα διάνυσμα εισόδου και ένα εξόδου (γνωρίζουμε το αποτέλεσμα). Ο αλγόριθμος, αναλύει τα δεδομένα εισόδου και παράγει ένα αποτέλεσμα, το οποίο συγκρίνει με το πραγματικό και προσπαθεί μέσα από διάφορες τροποποιήσεις να φτάσει κοντά σε αυτό. Αυτό απαιτεί

από τον αλγόριθμο να γενικεύει από τα δεδομένα εισόδου ώστε να μπορεί να προβλέπει το αποτέλεσμα χωρίς να το γνωρίζει, στη φάση ελέγχου.

Μη Επιβλεπόμενη μάθηση:

Η μη επιβλεπόμενη μάθηση(unsupervised learning), είναι η μέθοδος κατά την οποία το δίκτυο, δεν γνωρίζει εξ αρχής το αποτέλεσμα. Η μη επιβλεπόμενη μάθηση συνδέεται στενά με το πρόβλημα της εκτίμησης πυκνότητας στις στατιστικές. Ωστόσο, μάθηση χωρίς επίβλεψη περιλαμβάνει επίσης πολλές άλλες τεχνικές που επιδιώκουν να συνοψίσουν τα βασικά χαρακτηριστικά των δεδομένων. Πολλές μέθοδοι που χρησιμοποιούνται σε μη επιβλεπόμενη μάθηση βασίζονται σε μεθόδους εξόρυξης δεδομένων που χρησιμοποιούνται για την προεργασία δεδομένων.

Ενισχυτική μάθηση:

Η ενισχυτική μάθηση (reinforcement learning) στην επιστήμη των υπολογιστών, είναι ένας γενικός όρος που έχει δοθεί σε μια οικογένεια τεχνικών στις οποίες το σύστημα μάθησης, προσπαθεί να μάθει μέσα από την άμεση αλληλεπίδραση με το περιβάλλον

Η έννοια της ενισχυτικής μάθησης είναι εμπνευσμένη από τα αντίστοιχα ανάλογα της μάθησης με επιβράβευση και τιμωρία που συναντώνται ως μοντέλα μάθησης των έμβιων όντων. Σκοπός του συστήματος μάθησης είναι να μεγιστοποιήσει μια συνάρτηση του αριθμητικού σήματος *ενίσχυσης* (ανταμοιβή), για παράδειγμα την αναμενόμενη τιμή του σήματος ενίσχυσης στο επόμενο βήμα. Το σύστημα δεν καθοδηγείται από κάποιον εξωτερικό επιβλέποντα για το ποια ενέργεια θα πρέπει να ακολουθήσει αλλά πρέπει να ανακαλύψει μόνο του ποιες ενέργειες είναι αυτές που θα του αποφέρουν το μεγαλύτερο κέρδος.

Οι αλγόριθμοι που χρησιμοποιήσαμε στη Διπλωματική Εργασία για την Πρόβλεψη Αποτελεσμάτων Αγώνων NBA, χρησιμοποιούν αποκλειστικά επιβλεπόμενη μάθηση, αφού βασιστήκαμε πάνω στα παλιά αποτελέσματα τα οποία γνωρίζαμε. Στις περιπτώσεις που γνωρίζουμε το πραγματικό αποτέλεσμα, συνίσταται να χρησιμοποιείται επιβλεπόμενη μάθηση.

Κεφάλαιο 3

Αλγόριθμος Back-Propagation

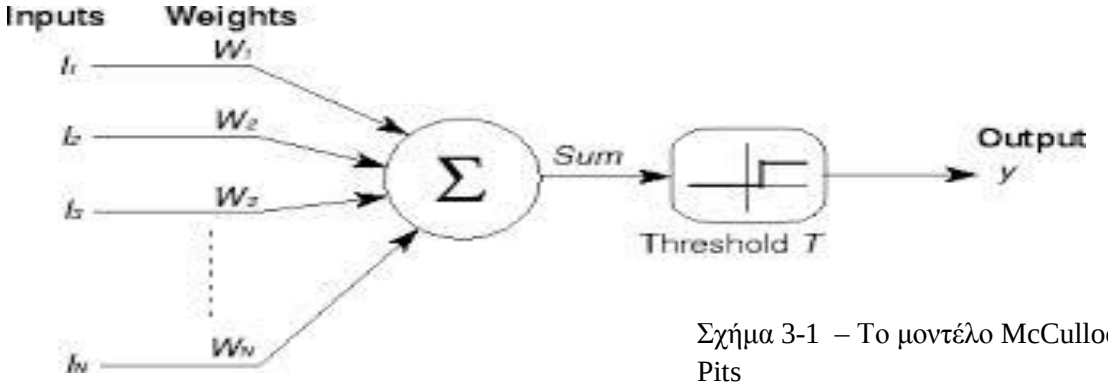
3.1 Εισαγωγή

Ο αλγόριθμος Back Propagation[8], αποτελεί τον πιο διαδεδομένο αλγόριθμο επιβλεπόμενης μάθησης. Σχεδιάστηκε το 1974 από τον Paul Werbos (Werbos,1974) [6] χωρίς όμως να λάβει αρκετή προσοχή και δημοσιότητα. Ακολούθως, οι David E. Rumelhart, Geoffrey E. Hinton και Ronald J. Williams, το 1976 (Rumelhart, Hinton, Williams, 1986) ασχολήθηκαν και τον έκαναν γνωστό στον ευρύτερο κόσμο των Νευρωνικών Δικτύων.

Είναι αποδοτικός στην εκπαίδευση feed-forward δικτύων, δηλαδή στα δίκτυα όπου τα αποτελέσματα από όλους τους νευρώνες πηγαίνουν στα επόμενα επίπεδα και όχι στα προηγούμενα κατά τη διάρκεια της εκπαίδευσης.

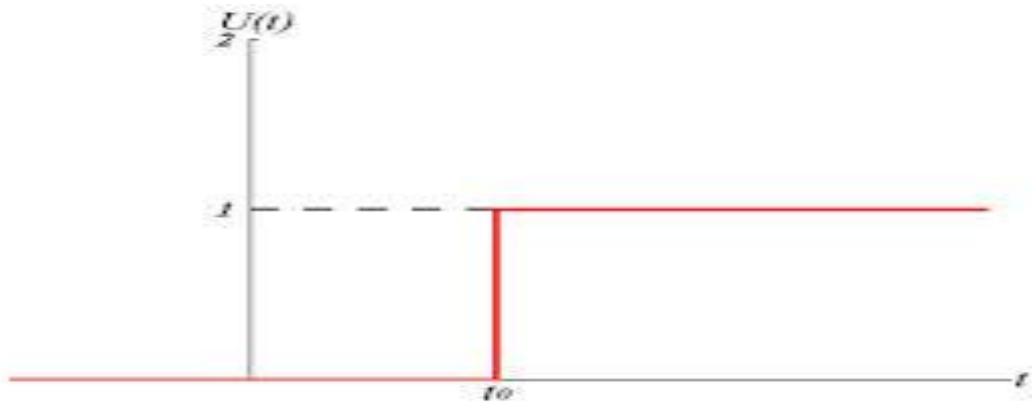
3.2 Το μοντέλο McCulloch-Pits

Ένα απλό μοντέλο πάνω στο επιβλεπόμενης μάθησης.



Σχήμα 3-1 – Το μοντέλο McCulloch-Pits

Παρατηρούμε στο σχήμα ότι έχουμε n εισόδους η κάθε μια από τις οποίες έχει μια τιμή. Αυτές οι είσοδοι, πολλαπλασιάζονται με τα αντίστοιχα βάρη. Το κάθε ένα από αυτά τα αποτελέσματα, προστίθεται έτσι ώστε να παράξουμε το άθροισμα όλων των εισόδων επί τα βάρη τους. Ο νευρώνας εξόδου, πυροβολεί 0 ή 1 ανάλογα αν το άθροισμα είναι μικρότερο ή μεγαλύτερο από ένα δοθέν κατώφλι. Αυτή είναι και η συνάρτηση ενεργοποίησης η οποία σ' αυτή την περίπτωση είναι η βηματοειδής.



Σχήμα 3.2 – Βηματοειδής συνάρτηση

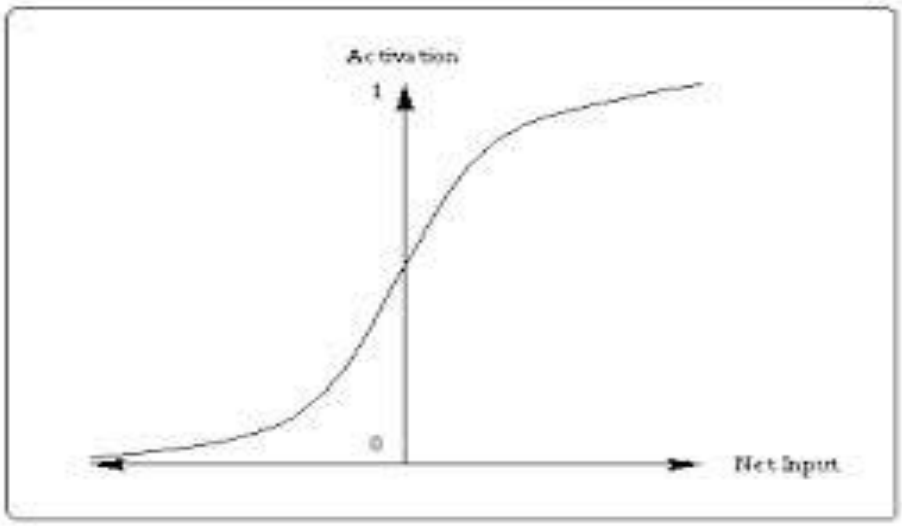
Τύπος υπολογισμού του αθροίσματος στο νευρώνα εξόδου

$$S = w_1.x_1 + w_2.x_2 + ... + w_n.x_n = \sum_{j=1}^{n} w_j.x_j$$

Αν το άθροισμα είναι κάτω από το κατώφλι το μ(S) είναι 0 (η συνάρτηση πυροβολεί 0) ενώ αν το άθροισμα είναι πάνω από το κατώφλι το μ(S) είναι 1 (η συνάρτηση πυροβολεί 1).

$$\mu\left(\sum_j w_{ij.}x_j(t)\right) = \begin{cases} 0 & if\left(\sum_j w_{ij.}x_j(t)\right) \le T \\ 1 & if\left(\sum_j w_{ij.}x_j(t)\right) \rangle T \end{cases}$$

Ο αλγόριθμος Back Propagation, δεν χρησιμοποιεί τη βηματοειδή συνάρτηση ως συνάρτηση ενεργοποίησης, αλλά τη σιγμοειδή.



Σχήμα 3.3 – Σιγμοειδής συνάρτηση

Ο λόγος που ο Back Propagation χρησιμοποιεί τη σιγμοειδή συνάρτηση, είναι γιατί είναι παραγωγίσιμη. Αυτό μας βοηθά στο να μην έχουμε δύο καταστάσεις (0 και 1) που απλά μας ενημερώνουν αν το άθροισμα μας είναι μικρότερο ή μεγαλύτερο από το κατώφλι, αλλά πλέον γνωρίζουμε και πόσο κοντά ή μακριά είμαστε σε αυτό. Έτσι μπορούμε βάση του δικτύου μας να προσαρμόσουμε κάποιους αριθμούς ανάλογα με την απόσταση μας από το κατώφλι.

Μαθηματικές εξισώσεις που χρησιμοποιεί ο Αλγόριθμος Back-Propagation

- $net_j = \sum_{\kappa} W_{jk}.O_k$

Τα βάρη από τον νευρώνα j στον νευρώνα k πολλαπλασιάζονται με την έξοδο του νευρώνα k

- $$O_j = \frac{1}{1+e^{(-a.net_j)}}$$

Το άθροισμα που υπολογίσαμε πιο πάνω, χρησιμοποιείται σ' αυτή την εξίσωση που είναι η σιγμοειδής συνάρτηση.

- $$\Delta w_{jk}(t) = \eta.\delta_j O_j + \lambda.\Delta w_{jk}(t-1)$$

$\Delta w_{jk}(t)$ είναι η αλλαγή στο βάρος W_{jk} στο χρόνο t, n είναι ο ρυθμός μάθησης του δικτύου και λ ο όρος ορμής.

- $$\delta_i = O_i(1-O_i)(t_i - O_i)$$

Ο υπολογισμός του λάθους στους νευρώνες εξόδου. O_i είναι το πραγματικό αποτέλεσμα εξόδου του νευρώνα ι και t_i το επιθυμητό αποτέλεσμα του.

- $$\delta_i = O_i(1-O_i)\sum_k W_{jk.\delta_k}$$

Ο υπολογισμός του λάθους στους νευρώνες του κρυφού επιπέδου. Εδώ δεν γνωρίζουμε το επιθυμητό αποτέλεσμα και έτσι ο όρος $t_i - O_i$ αντικαθίσταται από τον όρο $W_{jk.\delta_k}$ όπου δκ είναι το λάθος στο νευρώνα κ του επόμενου επιπέδου και W_{jk} το βάρος που ενώνει το νευρώνα κ με το νευρώνα ι.

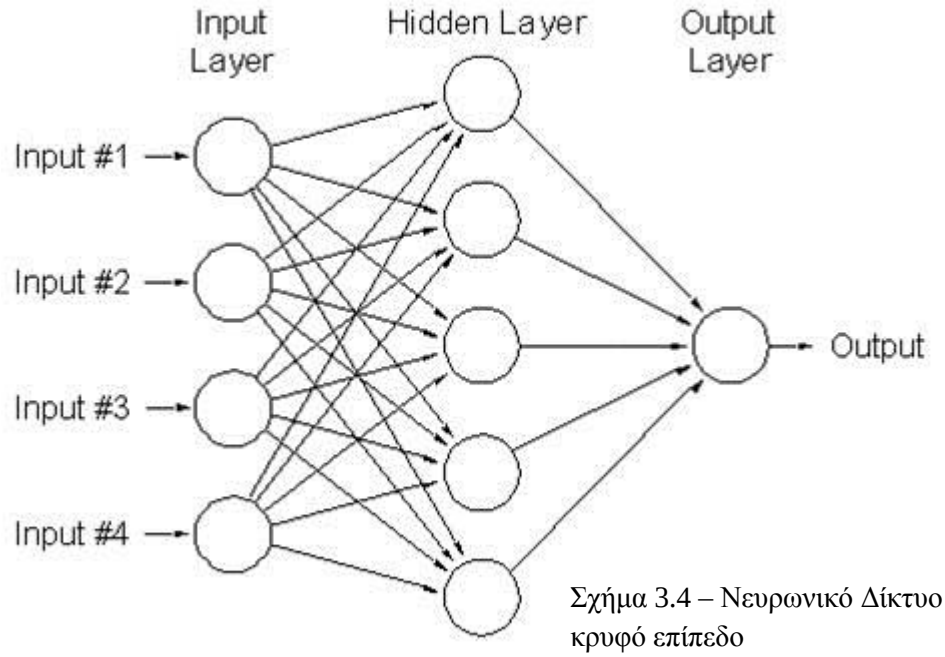
3.3 Εκτέλεση Αλγόριθμου

Δίνουμε στο νευρωνικό δίκτυο τα δεδομένα εκπαίδευσης και αρχικοποιούμε τα επιθυμητά αποτελέσματα στους νευρώνες εξόδου του δικτύου. Ακολούθως, το δίκτυο υπολογίζει τις πραγματικές τιμές στους νευρώνες εξόδου του δικτύου χρησιμοποιώντας τις ανάλογες εξισώσεις. Συγκρίνουμε τα πραγματικό αποτέλεσμα με το επιθυμητό στους νευρώνες εξόδου και υπολογίζουμε το σφάλμα. Τα λάθη

διαδίδονται προς τα πίσω με σκοπό την τροποποίηση των βαρών στους νευρώνες ώστε να μειώσουμε το σφάλμα. Επαναλαμβάνουμε αυτή τη διαδικασία μέχρι να μειώσουμε το σφάλμα του δικτύου αρκετά, για να βελτιωθούν τα ποσοστά σωστών προβλέψεών μας.

Παράδειγμα δικτύου

Σχήμα νευρωνικού δικτύου με αλγόριθμο Back Propagation.



Σχήμα 3.4 – Νευρωνικό Δίκτυο με ένα κρυφό επίπεδο

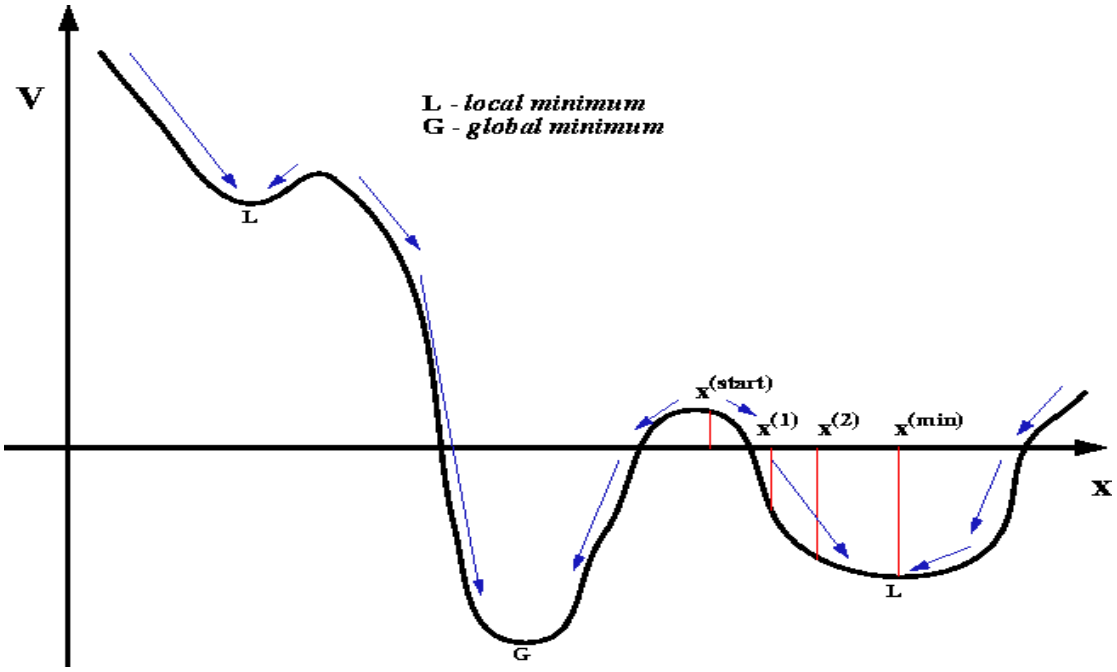
Παρατηρούμε ότι έχουμε το επίπεδο εισόδου όπου οι νευρώνες αρχικοποιούνται με τις τιμές των διανυσμάτων που θα χρησιμοποιήσουμε για εκπαίδευση του δικτύου. Ακολούθως, έχουμε το κρυφό επίπεδο το οποίο σ' αυτή την περίπτωση περιλαμβάνει ένα στρώμα (θα μπορούσε να περιλαμβάνει και δύο). Τέλος, έχουμε το επίπεδο εξόδου του δικτύου. Οι νευρώνες του κρυφού επιπέδου, ονομάζονται ενεργοί νευρώνες, αφού χρησιμοποιούν τη συνάρτηση ενεργοποίησης για να υπολογίσουν την έξοδο κάθε φορά. Όπως φαίνεται και στο σχήμα, όλοι οι κόμβοι, είναι πλήρως συνδεδεμένοι μεταξύ τους.

3.4 Πιθανά Προβλήματα

Κατά την εκπαίδευση ενός δικτύου με τον αλγόριθμο Back Propagation, μπορεί να παρατηρηθούν δύο προβλήματα.

Πρόβλημα Τοπικού Ελάχιστου

Χρησιμοποιώντας τη μέθοδο κατάβασης κλίσης, προσπαθούμε να βρούμε το χαμηλότερο σημείο της γραφικής παράστασης λάθους του δικτύου. Το χαμηλότερο σημείο αποτελεί το ολικό ελάχιστο. Υπάρχει πιθανότητα να εντοπίσουμε ένα άλλο, τοπικό ελάχιστο και να θεωρήσουμε ότι είναι αυτό το χαμηλότερο σημείο, ενώ στην πραγματικότητα δεν είναι.



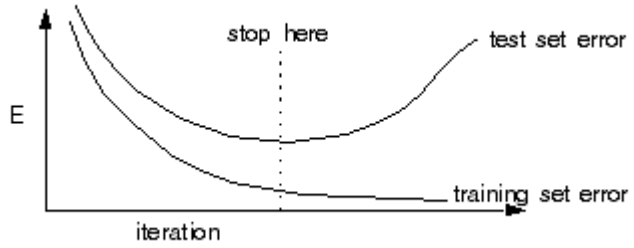
Σχήμα 3.5 – Τοπικό ελάχιστο, Ολικό ελάχιστο

Για να αντιμετωπιστεί αυτό το πρόβλημα, μπορούμε να μειώσουμε το ρυθμό μάθησης του δικτύου, να εισαγάγουμε τον όρο ορμής ο οποίος λαμβάνει υπόψη την αμέσως

προηγούμενη αλλαγή των βαρών που έγινε στο δίκτυο. Ακόμη, μπορούμε να προσθέσουμε περισσότερους κόμβους στο δίκτυο, έτσι ώστε να κωδικοποιούμε καλύτερα τα δεδομένα.

Πρόβλημα Υπερεκπαίδευσης

Σε αυτή την περίπτωση παρατηρείται το φαινόμενο, το δίκτυο από ένα σημείο και μετά αντί να βελτιώνεται, χειροτερεύει. Ο λόγος που συμβαίνει αυτό, είναι γιατί έχει εκπαιδευτεί πάρα πολύ και αντί να γενικεύει και να προσπαθεί να προβλέψει, απομνημονεύει τα πρότυπα εκπαίδευσης.



Σχήμα 3.6 – Πρόβλημα υπερεκπαίδευσης

Το πρόβλημα της υπερεκπαίδευσης μπορεί να αντιμετωπιστεί είτε με το να σταματήσουμε την εκπαίδευση γρήγορα, είτε με το να μειώσουμε την πολυπλοκότητα του δικτύου (να χρησιμοποιήσουμε λιγότερους κόμβους).

Κεφάλαιο 4

Δίκτυα RBF

4.1 Εισαγωγή στα δίκτυα RBF

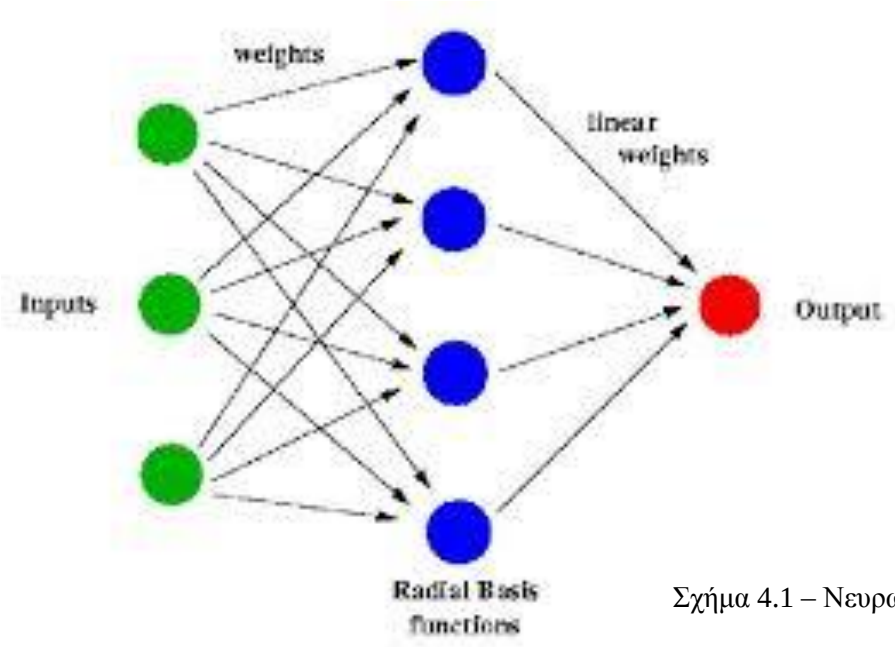
Το 1988 οι Broomhead και Lowe περιέγραψαν μία διαδικασία για το Σχεδιασμό FeedForward δικτύων (προς τα εμπρός τροφοδότηση) χρησιμοποιώντας συναρτήσεις ακτινικής βάσης(RBF).

Τα RBF δίκτυα, ανήκουν στην κατηγορία της επιβλεπόμενης μάθησης. Δηλαδή, γνωρίζουμε εξ αρχής το επιθυμητό αποτέλεσμα.

Τα RBF δίκτυα, αποτελούνται από ένα συνδεδεμένο δίκτυο νευρώνων, που βρίσκονται στο επίπεδο εισόδου, στο κρυφό επίπεδο και στο επίπεδο εξόδου. Οι είσοδοι του πρώτου επιπέδου, περνούν στους νευρώνες του κρυφού επιπέδου, τα λεγόμενα κέντρα. Στο κρυφό επίπεδο, γίνονται όλοι οι υπολογισμοί. Πρώτα υπολογίζεται η ευκλείδεια απόσταση που είναι η διαφορά του input vector με το center vector. Το αποτέλεσμα της ευκλείδειας απόστασης, περνά μέσα από την Γκαουσιανή συνάρτηση, ούτως ώστε να υπολογιστεί η έξοδος που στέλνεται στο επίπεδο εξόδου.

Το θεώρημα του Cover

Ο Cover το 1965 (Cover,1965), διατύπωσε ότι ένα πολύπλοκο πρόβλημα κατηγοριοποίησης προτύπων, είναι πιθανότερο να είναι γραμμικά διαχωρίσιμο όταν αποτιμηθεί μη γραμμικά σε διανυσματικό χώρο μεγαλύτερης διαστατικότητας, παρά σε διανυσματικό χώρο μικρότερης διαστατικότητας. Τα δίκτυα RBF στηρίζονται στο θεώρημα του Cover. Με τη χρήση της Γκαουσιανής συνάρτησης και με την αύξηση των νευρώνων του κρυφού δικτύου, επιτυγχάνουμε αυτή την αποτίμηση και έτσι μετατρέπουμε τα μη γραμμικά διαχωρίσιμα προβλήματα σε γραμμικά διαχωρίσιμα.



Σχήμα 4.1 – Νευρωνικό Δίκτυο RBF

4.2 Αρχιτεκτονική Δικτύων RBF

Με τη χρησιμοποίηση των RBF, προσπαθούμε να λύσουμε κάποιο πρόβλημα με μια λειτουργία προσέγγισης διαδικασίας (function approximation problem) [].

Η αρχιτεκτονική RBF είναι παρόμοια με αυτή των Πολυεπίπεδων Δικτύων Perceptron. Μάλιστα, στην περίπτωση που επιλέξουμε τις κατάλληλες παραμέτρους της RBF, τότε είναι σαν να χρησιμοποιούμε Μονοεπίπεδο Δίκτυο Perceptron, έτσι η ταχύτητα της διαδικασίας εκμάθησης του δικτύου μας αυξάνεται.

Τα δίκτυα έχουν συνήθως τρία στρώματα: ένα στρώμα εισόδου, ένα κρυφό στρώμα με μία μη γραμμική συνάρτηση ενεργοποίησης και ένα γραμμικό στρώμα εξόδου.

Η έξοδος του RBF μπορεί να εκφραστεί ως:

yi(p) = w0 + w1g1(x(p)) + w2g2(x(p)) + ……..+ gJ(x(p))

όπου το g είναι μια εξίσωση μη γραμμικού μετασχηματισμού στην οποία τα δεδομένα εισόδου συνεχώς μεταβάλλονται καθώς διαδίδονται από το στρώμα εισόδου στο κρυφό στρώμα.

4.3 Μέθοδοι Εκπαίδευσης RBF δικτύων

Τα RBF δίκτυα, μπορούν να εκπαιδευτούν με δύο μεθόδους, αυτή των σταθερών και αυτή των μεταβλητών κέντρων. Στη μέθοδο σταθερών κέντρων, κατά τη διάρκεια της εκπαίδευσης τα κέντρα δεν αλλάζουν, όμως μεταβάλλονται οι συντελεστές. Είναι σημαντικό να γίνει σωστή επιλογή των κέντρων, καθώς επίσης και σωστή επιλογή του αριθμού τους. Δεν μπορούμε να ξέρουμε εξ αρχής τον αριθμό όμως μπορούμε με διάφορες δοκιμές να τον καθορίσουμε. Στη μέθοδο μεταβλητών κέντρων, χρησιμοποιούμε ένα στοχαστικό αλγόριθμο ταχυτέρας καθόδου. Τα κέντρα αλλάζουν και αυτό στηρίζεται στη μέθοδο κατάβασης κλίσης.

4.3.1 Εκπαίδευση με σταθερά κέντρα:

Επιλέγουμε τα κατάλληλα κέντρα R, τα οποία θα μείνουν σταθερά κατά την διάρκεια της εκπαίδευσης. Στη συνέχεια επιλέγουμε τη διασπορά.

Ακολούθως, αρχικοποιούμε τα βάρη c (coefficients), με μικρές τυχαίες τιμές. Οι συντελεστές μεταβάλλονται κατά τη διάρκεια της εκπαίδευσης.

Υπολογίζουμε την έξοδο και βρίσκουμε τα βέλτιστα βάρη.

Η έξοδος του δικτύου δίνεται από την εξίσωση :

$$y = \sum_{i=1}^{n} c_i \, \Phi(\|x - R_i\|)$$

Όπου c_i είναι ο συντελεστής του κέντρου i.

$$\Phi(\|x - R_i\|)$$

Είναι το αποτέλεσμα της μη γραμμικής συνάρτησης φ (συνήθως αυτή την συνάρτηση την αποτελεί η Γκαουσιανή συνάρτηση), στη διαφορά του διανύσματος εισόδου με το διάνυσμα του κέντρου i.

Η μέθοδος των σταθερών κέντρων, χρησιμοποιείται σε προβλήματα, τα οποία χρειάζονται ακρίβεια. Στις περιπτώσεις δηλαδή που έχουμε παρεμβολή.

4.3.2 Εκπαίδευση με μεταβλητά κέντρα:

Η μέθοδος εκμάθησης με μεταβλητά κέντρα στηρίζεται στη μέθοδο κατάβασης κλίσης.

Επιλέγουμε τα κατάλληλα κέντρα R και τα κατάλληλα σ. Αρχικοποιούμε τα βάρη c με μικρές τυχαίες τιμές. Δίνουμε στο δίκτυο ένα διανυσματικό δείγμα και βρίσκουμε την έξοδο.

Αφού υπολογίσουμε την πραγματική έξοδο, με τη μέθοδο κατάβασης κλίσης, βρίσκουμε τις νέες τιμές για c, R και σ.

Αν το δίκτυο δείξει ικανοποιητική σύγκλιση, τότε η διαδικασία εκπαίδευσης σταματά.

Το στιγμιαίο συνολικό φάσμα δίδεται από την εξίσωση:

$$J[k] = \frac{1}{2}[e[k]]^2 = \frac{1}{2}\left\{d[k] - \sum_{k=1}^{n} c_k[k] * \Phi(x[k], R_k[k])\right\}^2$$

Όπου:

$$d[k]$$

Είναι το πραγματικό αποτέλεσμα

Και

$$\sum_{k=1}^{n} c_k[k] * \Phi(x[k], R_k[k])$$

Είναι το επιθυμητό αποτέλεσμα.

Η μέθοδος των μεταβλητών κέντρων, χρησιμοποιείται σε προβλήματα, τα οποία έχουν πολλά δεδομένα και έτσι θέλουμε το δίκτυο να τα μάθει κατά προσέγγιση και όχι με παρεμβολή.

Κεφάλαιο 5

Δεδομένα Εισόδου

5.1 Δεδομένα Εισόδου του Δικτύου

Οι δύο αλγόριθμοι που υλοποιήθηκαν (Back Propagation και RBF), όπως έγινε εμφανές δέχονται κάποια δεδομένα εισόδου, τα επεξεργάζονται και προσπαθούν να προβλέψουν το τελικό αποτέλεσμα ενός επερχόμενου αγώνα.

Ασφαλώς τα δεδομένα στα οποία βασιζόμαστε, είναι τα αποτελέσματα των παλαιότερων αγώνων. Έχει σημασία όμως το τι αντλούμε από αυτά τα αποτελέσματα και τι ακριβώς από αυτά δίνουμε σαν είσοδο στους αλγόριθμούς μας.

Κάπου εδώ πρέπει να αναφέρω ότι απέφυγα να χρησιμοποιήσω δεδομένα τύπου απόδοσης καλαθοσφαιριστών, απουσίες καλαθοσφαιριστών, ουδέτερο γήπεδο, κίνητρο, διαιτητές. Ο λόγος που απέφυγα τη χρησιμοποίηση όλων αυτών είναι γιατί τα δίκτυα θα γίνονταν πολύ πολύπλοκα. Η εκπαίδευση τους θα γινόταν πιο δύσκολη και ίσως κάποια ορθά αποτελέσματα που έδιναν, να ήταν λόγω τύχης.

Και στους δύο αλγόριθμους δοκιμάσαμε να χρησιμοποιήσουμε το εξής διάνυσμα εισόδου για κάθε αγώνα.

i. **Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της**
   Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως γηπεδούχος, τον μέσο όρο πόντων που πέτυχε αγωνιζόμενη στην έδρα της, μέχρι τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

ii. Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της

Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως γηπεδούχος, τον μέσο όρο πόντων που δέχτηκε αγωνιζόμενη στην έδρα της, μέχρι τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

iii. **Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας**
Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως φιλοξενούμενη, τον μέσο όρο πόντων που πέτυχε αγωνιζόμενη εκτός έδρας, μέχρι τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

iv. **Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας**
Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως φιλοξενούμενη, τον μέσο όρο πόντων που δέχτηκε αγωνιζόμενη εκτός έδρας, μέχρι τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

v. **Ποσοστό παιχνιδιών της γηπεδούχου ομάδας στην έδρα της, στα οποία σημειώθηκε πάνω από ένας συγκεκριμένος αριθμός πόντων.**
Ανάλογα με το κατώφλι που εμείς επιλέγουμε να δώσουμε στο δίκτυο (π.χ 190, 200, 210 πόντοι) υπολογίζουμε το ποσοστό των παιχνιδιών στα οποία σημειώθηκαν περισσότεροι πόντοι από αυτό, όταν αγωνίστηκε η γηπεδούχος ομάδα στην έδρα της.

vi. **Ποσοστό παιχνιδιών της φιλοξενούμενης ομάδας εκτός έδρας, στα οποία σημειώθηκε πάνω από ένας συγκεκριμένος αριθμός πόντων.**
Ανάλογα με το κατώφλι που εμείς επιλέγουμε να δώσουμε στο δίκτυο (π.χ 190, 200, 210 πόντοι) υπολογίζουμε το ποσοστό των παιχνιδιών στα οποία σημειώθηκαν περισσότεροι πόντοι από αυτό, όταν αγωνίστηκε η φιλοξενούμενη ομάδα εκτός έδρας.

vii. **Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες**
Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως γηπεδούχος, τον μέσο όρο πόντων που πέτυχε αγωνιζόμενη στην έδρα της, τους τελευταίους έξι αγώνες

πριν τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

viii. **Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες**

Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως γηπεδούχος, τον μέσο όρο πόντων που δέχτηκε αγωνιζόμενη στην έδρα της, τους τελευταίους έξι αγώνες πριν τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

ix. Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες

Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως φιλοξενούμενη, τον μέσο όρο πόντων που πέτυχε αγωνιζόμενη εκτός έδρας, τους τελευταίους έξι αγώνες πριν τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

x. Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες

Υπολογίζουμε για κάθε ομάδα που αγωνίζεται ως φιλοξενούμενη, τον μέσο όρο πόντων που δέχτηκε αγωνιζόμενη εκτός έδρας, τους τελευταίους έξι αγώνες πριν τον αγώνα που επίκειται, του οποίου θέλουμε να προβλέψουμε το αποτέλεσμα.

xi. Ποσοστό παιχνιδιών της γηπεδούχου ομάδας στην έδρα της, στα οποία σημειώθηκε πάνω από ένας συγκεκριμένος αριθμός πόντων, τους τελευταίους έξι αγώνες.

Ανάλογα με το κατώφλι που εμείς επιλέγουμε να δώσουμε στο δίκτυο (π.χ 190, 200, 210 πόντοι) υπολογίζουμε το ποσοστό των παιχνιδιών στα οποία σημειώθηκαν περισσότεροι πόντοι από αυτό, όταν αγωνίστηκε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες.

xii. **Ποσοστό παιχνιδιών της φιλοξενούμενης ομάδας εκτός έδρας, στα οποία σημειώθηκε πάνω από ένας συγκεκριμένος αριθμός πόντων, τους τελευταίους έξι αγώνες.**

Ανάλογα με το κατώφλι που εμείς επιλέγουμε να δώσουμε στο δίκτυο (π.χ 190, 200, 210 πόντοι) υπολογίζουμε το ποσοστό των παιχνιδιών στα οποία σημειώθηκαν περισσότεροι πόντοι από αυτό, όταν αγωνίστηκε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες.

5.2 Επεξεργασία Δεδομένων

Τα δεδομένα που χρησιμοποιήθηκαν για την εκπαίδευση του κάθε δικτύου, ήταν τα αποτελέσματα των αγωνιστικών περιόδων από το 2003 μέχρι το 2012.
Στην εκπαίδευση του δικτύου, επέλεξα να λαμβάνω υπόψη όλα τα προηγούμενα αποτελέσματα των ομάδων, χωρίς να ξεχωρίζω τις αγωνιστικές περιόδους. Έγινε έλεγχος κατά πόσο οι σωστές προβλέψεις θα ήταν περισσότερες αν ξεχώριζα τις περιόδους, χωρίς όμως να φανεί κάτι τέτοιο.

Τα δεδομένα που χρησιμοποίησα, τα άντλησα από τη σελίδα www.basketball-reference.com.

www.basketball-reference.com/leagues/NBA_2010_games.html

NBA Seasons ▾ | Summary ▾ | Standings | Schedule & Results | Leaders | Stats ▾ | Other ▾

Tue, Oct 27, 2009	Box Score	Boston Celtics	95	Cleveland Cavaliers	89		
Tue, Oct 27, 2009	Box Score	Washington Wizards	102	Dallas Mavericks	91		
Tue, Oct 27, 2009	Box Score	Los Angeles Clippers	92	Los Angeles Lakers	99		
Tue, Oct 27, 2009	Box Score	Houston Rockets	87	Portland Trail Blazers	96		
Wed, Oct 28, 2009	Box Score	Indiana Pacers	109	Atlanta Hawks	120		
Wed, Oct 28, 2009	Box Score	Charlotte Bobcats	59	Boston Celtics	92		
Wed, Oct 28, 2009	Box Score	Utah Jazz	105	Denver Nuggets	114		
Wed, Oct 28, 2009	Box Score	Houston Rockets	108	Golden State Warriors	107		
Wed, Oct 28, 2009	Box Score	Phoenix Suns	109	Los Angeles Clippers	107		
Wed, Oct 28, 2009	Box Score	Detroit Pistons	96	Memphis Grizzlies	74		
Wed, Oct 28, 2009	Box Score	New York Knicks	93	Miami Heat	115		
Wed, Oct 28, 2009	Box Score	New Jersey Nets	93	Minnesota Timberwolves	95		
Wed, Oct 28, 2009	Box Score	Sacramento Kings	89	Oklahoma City Thunder	102		
Wed, Oct 28, 2009	Box Score	Philadelphia 76ers	106	Orlando Magic	120		
Wed, Oct 28, 2009	Box Score	New Orleans Hornets	96	San Antonio Spurs	113		
Wed, Oct 28, 2009	Box Score	Cleveland Cavaliers	91	Toronto Raptors	101		
Thu, Oct 29, 2009	Box Score	San Antonio Spurs	85	Chicago Bulls	92		
Thu, Oct 29, 2009	Box Score	Denver Nuggets	97	Portland Trail Blazers	94		
Fri, Oct 30, 2009	Box Score	Washington Wizards	89	Atlanta Hawks	100		
Fri, Oct 30, 2009	Box Score	Chicago Bulls	90	Boston Celtics	118		
Fri, Oct 30, 2009	Box Score	New York Knicks	100	Charlotte Bobcats	102	2OT	
Fri, Oct 30, 2009	Box Score	Oklahoma City Thunder	91	Detroit Pistons	83		
Fri, Oct 30, 2009	Box Score	Miami Heat	96	Indiana Pacers	83		
Fri, Oct 30, 2009	Box Score	Dallas Mavericks	94	Los Angeles Lakers	80		
Fri, Oct 30, 2009	Box Score	Toronto Raptors	107	Memphis Grizzlies	115		
Fri, Oct 30, 2009	Box Score	Cleveland Cavaliers	104	Minnesota Timberwolves	87		
Fri, Oct 30, 2009	Box Score	Orlando Magic	95	New Jersey Nets	85		
Fri, Oct 30, 2009	Box Score	Sacramento Kings	92	New Orleans Hornets	97		
Fri, Oct 30, 2009	Box Score	Milwaukee Bucks	86	Philadelphia 76ers	99		
Fri, Oct 30, 2009	Box Score	Golden State Warriors	101	Phoenix Suns	123		
Fri, Oct 30, 2009	Box Score	Los Angeles Clippers	98	Utah Jazz	111		
Sat, Oct 31, 2009	Box Score	Charlotte Bobcats	79	Cleveland Cavaliers	90		
Sat, Oct 31, 2009	Box Score	Portland Trail Blazers	107	Houston Rockets	111		
Sat, Oct 31, 2009	Box Score	Dallas Mavericks	93	Los Angeles Clippers	84		
Sat, Oct 31, 2009	Box Score	Detroit Pistons	85	Milwaukee Bucks	96		
Sat, Oct 31, 2009	Box Score	Philadelphia 76ers	141	New York Knicks	127	OT	
Sat, Oct 31, 2009	Box Score	Sacramento Kings	94	San Antonio Spurs	113		
Sat, Oct 31, 2009	Box Score	New Jersey Nets	104	Washington Wizards	123		
Sun, Nov 1, 2009	Box Score	New Orleans Hornets	87	Boston Celtics	97		
Sun, Nov 1, 2009	Box Score	Memphis Grizzlies	123	Denver Nuggets	133		
Sun, Nov 1, 2009	Box Score	Atlanta Hawks	110	Los Angeles Lakers	118		
Sun, Nov 1, 2009	Box Score	Chicago Bulls	87	Miami Heat	95		
Sun, Nov 1, 2009	Box Score	Portland Trail Blazers	83	Oklahoma City Thunder	74		
Sun, Nov 1, 2009	Box Score	Minnesota Timberwolves	112	Phoenix Suns	120		
Sun, Nov 1, 2009	Box Score	Orlando Magic	125	Toronto Raptors	116		
Mon, Nov 2, 2009	Box Score	New Jersey Nets	68	Charlotte Bobcats	79		
Mon, Nov 2, 2009	Box Score	Minnesota Timberwolves	90	Los Angeles Clippers	93		

Σχήμα 5.1 – Αποτελέσματα αγώνων για παλαιότερες σεζόν

Όπως μπορείτε να δείτε στην εικόνα, η σελίδα παρέχει τα αποτελέσματα για κάθε σεζόν σε αυτή τη μορφή. Αρχικά η ημερομηνία, ακολούθως η ομάδα που αγωνίζεται εκτός έδρας, μετά ο αριθμός των πόντων που πέτυχε. Στη συνέχεια, η ομάδα που αγωνίζεται εντός έδρας και ο αριθμός των πόντων που πέτυχε. Εγώ, στο δίκτυο, δίνω πρώτα την ομάδα που αγωνίστηκε εντός έδρας και μετά την ομάδα που αγωνίστηκε εκτός, έτσι αντίστρεψα τις στήλες στα δεδομένα εισόδου μου.

Σε αυτό το σημείο πρέπει να πω ότι στην τελευταία στήλη, αναγράφεται η συντομογραφία OT (OverTime) για τους αγώνες που οδηγήθηκαν στην παράταση. Επίσης, το 2OT σημαίνει ότι ο αγώνας είχε 2 πεντάλεπτα παράταση, αφού με το τέλος της πρώτης, οι ομάδες εξακολουθούσαν να ήταν ισόπαλες. Όπου υπήρχε

παράταση, επεξεργάστηκα το αποτέλεσμα έτσι ώστε τελικό να ήταν αυτό πριν τις παρατάσεις. Ο λόγος που αυτό ήταν απαραίτητο, είναι γιατί το σκορ στις παρατάσεις μπορεί να εκτοξευθεί σε τέτοιο βαθμό, που να είναι εξωπραγματικό για σκορ κανονικής διάρκειας ενός αγώνα, έτσι θα μπορούσε να επηρεάσει αρνητικά το δίκτυο.

Τα αρχεία που χρησιμοποίησα για κάθε σεζόν, ήταν ένα που περιείχε όλα τα αποτελέσματα της, και ακόμα 30 ένα για κάθε ομάδα. Μέσω της κλάσης, FileCreator.java περνούσα το file που περιείχε όλα τα αποτελέσματα και δημιουργούσα ένα file για κάθε ομάδα και τα αποτελέσματά της.

Παράδειγμα αρχείου ολόκληρης χρονιάς (δεν είναι ολόκληρο) :

DALLAS SACRAMENTO 107 98
DETROIT HOUSTON 87 79
LALAKERS DENVER 89 78
BOSTON PHILADELPHIA 95 98
CLEVELANDINDIANA 85 85
GOLDENSTATE PORTLAND 75 78
LACLIPPERS SEATTLE 114 84
MEMPHIS WASHINGTON 91 103
MINNESOTA NEWYORK 99 93
NJNETS MIAMI 77 100
NEWORLEANS DALLAS 91 106
ORLANDO MILWAUKEE 93 92
PHOENIX ATLANTA 112 82
SANANTONIO SACRAMENTO 101 85
TORONTO HOUSTON 95 88
UTAH LALAKERS 104 78
CHARLOTTE WASHINGTON 96 103
DENVER MINNESOTA 88 88
MIAMI CLEVELAND 92 86
BOSTON INDIANA 94 100
CHICAGO NJNETS 94 94
GOLDENSTATE UTAH 80 102

LALAKERS	SANANTONIO	96	105
MEMPHIS	HOUSTON	81	89
NEWORLEANS	ORLANDO	89	90
PHILADELPHIA	PHOENIX	98	108
PORTLAND	LACLIPPERS	94	81
SEATTLE	ATLANTA	106	85
TORONTO	DETROIT	101	89
CHARLOTTE	ORLANDO	111	100
DALLAS	MEMPHIS	112	88
DENVER	UTAH	82	106
DETROIT	PHILADELPHIA	99	91
GOLDENSTATE	LACLIPPERS	86	94
HOUSTON	SACRAMENTO	93	93
INDIANA	CHICAGO	100	90
MILWAUKEE	CLEVELAND	102	88
MINNESOTA	NEWORLEANS	99	92
NJNETS	PHOENIX	80	112
NEWYORK	BOSTON	73	107
WASHINGTON	MIAMI	106	118
LALAKERS	ATLANTA	106	90
SEATTLE	SANANTONIO	113	94
TORONTO	PORTLAND	101	97
DALLAS	GOLDENSTATE	88	88
LACLIPPERS	DETROIT	83	83
UTAH	DENVER	102	91
ATLANTA	CLEVELAND	79	93
CHICAGO	PHOENIX	74	94
DENVER	SEATTLE	88	108
HOUSTON	MEMPHIS	90	87
MIAMI	WASHINGTON	103	93
MINNESOTA	INDIANA	101	102
NJNETS	PORTLAND	64	60
NEWORLEANS	LALAKERS	98	106
NEWYORK	PHILADELPHIA	96	88

ORLANDO DALLAS 94 84
SACRAMENTO TORONTO 108 92
BOSTON PORTLAND 90 88
CLEVELANDPHOENIX 99 99
INDIANA LACLIPPERS 68 102
MEMPHIS LALAKERS 110 87
MILWAUKEE CHARLOTTE 102 100
PHILADELPHIA NJNETS 96 96
SANANTONIO GOLDENSTATE 91 71
SEATTLE SACRAMENTO 108 78
UTAH TORONTO 95 104
WASHINGTON ORLANDO 106 96
DENVER DETROIT 117 109
HOUSTON MINNESOTA 91 96
MIAMI DALLAS 93 113
BOSTON CHARLOTTE 91 74
MEMPHIS GOLDENSTATE 96 67
NEWORLEANS ATLANTA 95 96
NEWYORK LACLIPPERS 110 96
ORLANDO LALAKERS 122 113
PHILADELPHIA INDIANA 100 100
SANANTONIO MIAMI 93 84

Παράδειγμα αρχείου ομάδας:

PHOENIX ATLANTA 112 82
SEATTLE ATLANTA 106 85
LALAKERS ATLANTA 106 90
ATLANTA CLEVELAND 79 93
NEWORLEANS ATLANTA 95 96
ATLANTA SANANTONIO 88 103
ATLANTA HOUSTON 88 84

INDIANA ATLANTA 93 86
ATLANTA UTAH 79 92
NEWYORK ATLANTA 104 88
ATLANTA MIAMI 93 99
ATLANTA ORLANDO 99 117
CHARLOTTE ATLANTA 107 92
ATLANTA NEWYORK 98 98
ATLANTA WASHINGTON 90 114
NJNETS ATLANTA 109 88
ATLANTA PHILADELPHIA 96 92
ATLANTA MEMPHIS 89 97
DETROIT ATLANTA 72 88
ATLANTA NJNETS 90 95
ATLANTA INDIANA 91 91
HOUSTON ATLANTA 92 69
ATLANTA PORTLAND 84 100
DALLAS ATLANTA 90 68
ATLANTA DALLAS 113 100
MIAMI ATLANTA 116 102
ATLANTA CLEVELAND 102 111
ATLANTA SEATTLE 79 94
WASHINGTON ATLANTA 104 101
CLEVELAND ATLANTA 101 85
ATLANTA SACRAMENTO 97 100
ATLANTA MILWAUKEE 103 80
BOSTON ATLANTA 106 94
ATLANTA CHARLOTTE 103 95
ATLANTA NJNETS 84 85
MIAMI ATLANTA 111 92
CHICAGO ATLANTA 95 85
ATLANTA BOSTON 100 96
ATLANTA CHICAGO 82 107
MINNESOTA ATLANTA 104 87
ATLANTA MIAMI 96 106

MEMPHIS ATLANTA 84 83
ATLANTA ORLANDO 80 79
DETROIT ATLANTA 99 84
PHILADELPHIA ATLANTA 103 85
ATLANTA INDIANA 79 84
ATLANTA LALAKERS 114 108
ORLANDO ATLANTA 101 96
MILWAUKEE ATLANTA 113 83
ATLANTA DENVER 96 100
CLEVELAND ATLANTA 111 89
SACRAMENTO ATLANTA 114 104
GOLDENSTATE ATLANTA 101 96
PORTLAND ATLANTA 102 101
LACLIPPERS ATLANTA 111 104
DENVER ATLANTA 97 74
UTAH ATLANTA 96 74
ATLANTA PHILADELPHIA 97 98
MILWAUKEE ATLANTA 105 101
BOSTON ATLANTA 95 91
TORONTO ATLANTA 101 101
ATLANTA GOLDENSTATE 92 105
ATLANTA DETROIT 100 100
ATLANTA WASHINGTON 93 122
ATLANTA NEWYORK 92 106
CHICAGO ATLANTA 105 91
ATLANTA PHOENIX 94 105
SANANTONIO ATLANTA 111 95
ATLANTA TORONTO 104 109
ORLANDO ATLANTA 109 102
WASHINGTON ATLANTA 102 99
ATLANTA BOSTON 100 116
ATLANTA LACLIPPERS 91 111
ATLANTA NEWORLEANS 86 96
TORONTO ATLANTA 96 96

ATLANTA MINNESOTA 105 98
ATLANTA CHARLOTTE 101 101
CHARLOTTE ATLANTA 105 84
ATLANTA CHICAGO 105 114
NEWYORK ATLANTA 118 118
ATLANTA DETROIT 68 95
PHILADELPHIA ATLANTA 110 86

5.3 Κανονικοποίηση Τιμών

Όπως προανέφερα, στους αλγόριθμους των Νευρωνικών Δικτύων, είναι σημαντική η κανονικοποίηση τιμών.
Στο διάνυσμα που δίνουμε σαν είσοδο, υπάρχουν τιμές που οι τιμές τους είναι πραγματικές τιμές πόντων, δηλαδή μπορεί να είναι και 90 και 100 και 120. Επίσης, υπάρχουν τιμές που αντιπροσωπεύουν ποσοστά, δηλαδή κυμαίνονται από το 0 μέχρι το 1. Αυτές οι μεγάλες διαφορές μεταξύ των τιμών πρέπει να αποφεύγονται στην εκπαίδευση των δικτύων, αφού μπορεί να προκαλέσουν ανεπιθύμητα αποτελέσματα. Με τη μέθοδο της κανονικοποίησης μπορούμε να αποφύγουμε αυτά τα προβλήματα.

Η εξίσωση που χρησιμοποιούμε είναι :

$$y = \frac{X - x_{min}}{x_{max} - x_{min}}$$

Το X αποτελεί την αρχική τιμή πριν την κανονικοποίηση, το Xmax τη μεγαλύτερη πιθανή τιμή για τη συγκεκριμένη μεταβλητή, το Xmin τη μικρότερη πιθανή τιμή για τη συγκεκριμένη μεταβλητή και το Y τη νέα τιμή της μεταβλητής μετά την κανονικοποίηση.

Με αυτό τον τρόπο, όλες οι διαστάσεις του διανύσματος έχουν τιμές από 0 μέχρι 1 και έτσι αποφεύγονται οι μεγάλες διαφορές, έτσι το δίκτυο εκπαιδεύεται χωρίς επιπλέον προβλήματα.

Κεφάλαιο 6

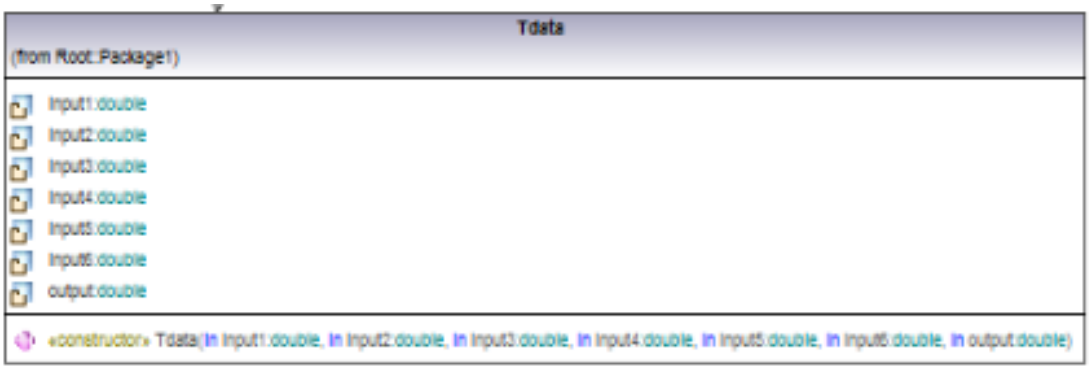
Σχεδιασμός και Υλοποίηση

6.1 Σχεδιασμός και υλοποίηση αλγόριθμου Back-Propagation

Στο σημείο αυτό, θα γίνει αναλυτική περιγραφή για το σχεδιασμό και την υλοποίηση του δικτύου που κάνει χρήση του αλγόριθμου Back-Propagation. Χρησιμοποίησα γλώσσα προγραμματισμού Java και το πρόγραμμα βασίζεται σε αντικειμενοστραφή προγραμματισμό.

Πιο κάτω περιγράφεται η κάθε κλάση ξεχωριστά, καθώς και οι ιδιότητές της.

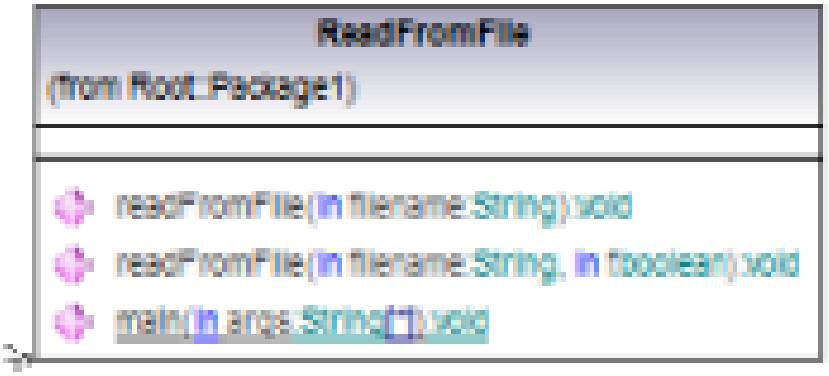
6.1.1 Κλάση Tdata



Η Κλάση αναπαριστά το training και test data. Αποτελείται από τα έξι inputs που υπολογίζουμε για να περάσουμε σαν είσοδο στον αλγόριθμο καθώς και το πραγματικό αποτέλεσμα που θα χρησιμοποιηθεί για να το συγκρίνουμε με αυτό που

θα παράξει ο αλγόριθμός μας. Σαν μέθοδο περιέχει μόνο τον κατασκευαστή για ένα αντικείμενο TData.

6.1.2 Κλάση ReadFromFile



Η κλάση ReadFromFile, περιέχει μεθόδους για να διαβάσουμε δύο τύπους αρχείων. Ο ένας τύπος αφορά το αρχείο που περιέχει τις παραμέτρους που θα χρησιμοποιήσουμε στον αλγόριθμο, όπως αριθμό επιπέδων, learning rate, sigma κλπ.

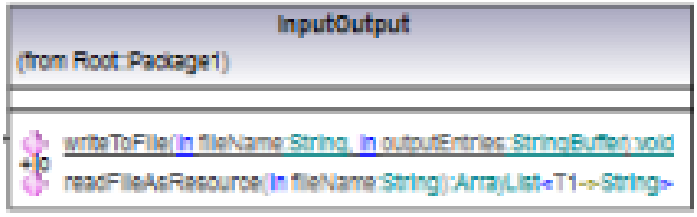
Ο άλλος τύπος αρχείου είναι το training και test file, που περιέχει τα inputs του αλγόριθμου που περιγράψαμε πιο πάνω, καθώς και το πραγματικό αποτέλεσμα ενός αγώνα.

6.1.3 Κλάση PrintInFile

Κλάση η οποία περιέχει μεθόδους για εκτύπωση αρχείων.

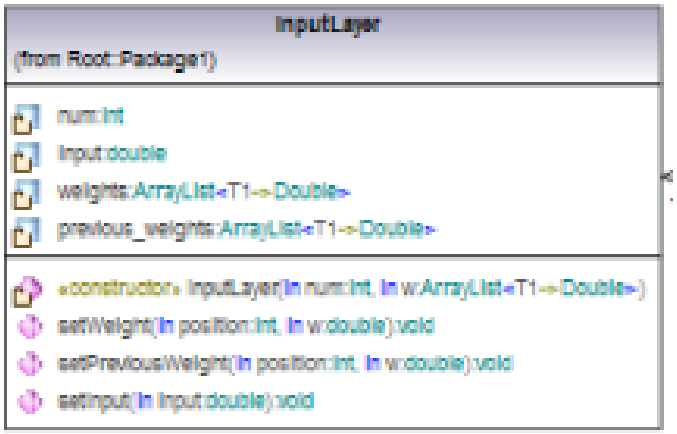
Η πρώτη μέθοδος εκτυπώνει ένα αρχείο που περιέχει τα βάρη στους νευρώνες στο κάθε επίπεδο, ενώ η δεύτερη εκτυπώνει ένα αρχείο που μας δείχνει το σφάλμα εκπαίδευσης, το οποίο με τις επαναλήψεις μειώνεται.

6.1.4 Κλάση InputOutput



Η κλάση InputOutput, χρησιμοποιείται για να τυπωθούν τα αρχεία που περιέχουν τα δεδομένα εκπαίδευσης και τα δεδομένα ελέγχου. Περιέχει μεθόδους που γράφουν ένα StringBuffer σε αρχείο και που διαβάζουν ένα αρχείο και παράγουν μία λίστα.

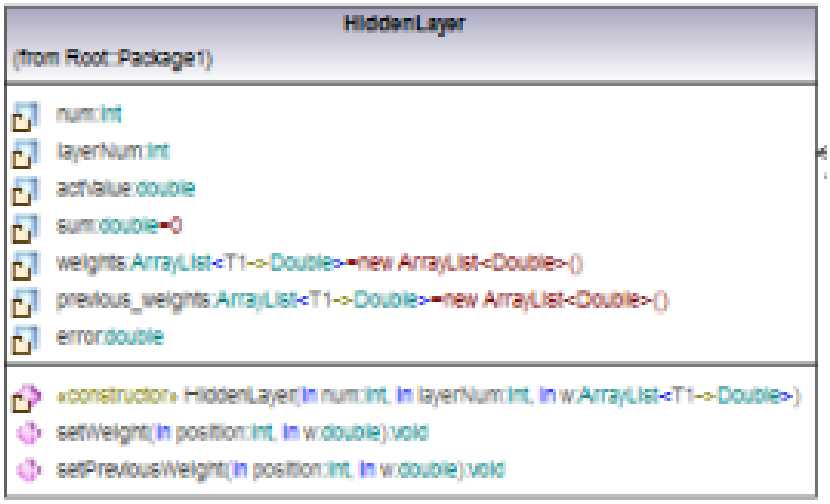
6.1.5 Κλάση InputLayer



Η κλάση InputLayer, αναπαριστά ένα νευρώνα που βρίσκεται στο επίπεδο εισόδου. Περιέχει τον αριθμό του νευρώνα, την τιμή που θα πάρει, καθώς και δύο λίστες, μια για τα βάρη και μια για τα προηγούμενα βάρη που είχε.

Ακόμη, περιέχει τον κατασκευαστή της κλάσης ενώ υπάρχουν οι μέθοδοι για να ορίσουμε τα βάρη, τα προηγούμενα βάρη και την τιμή του.

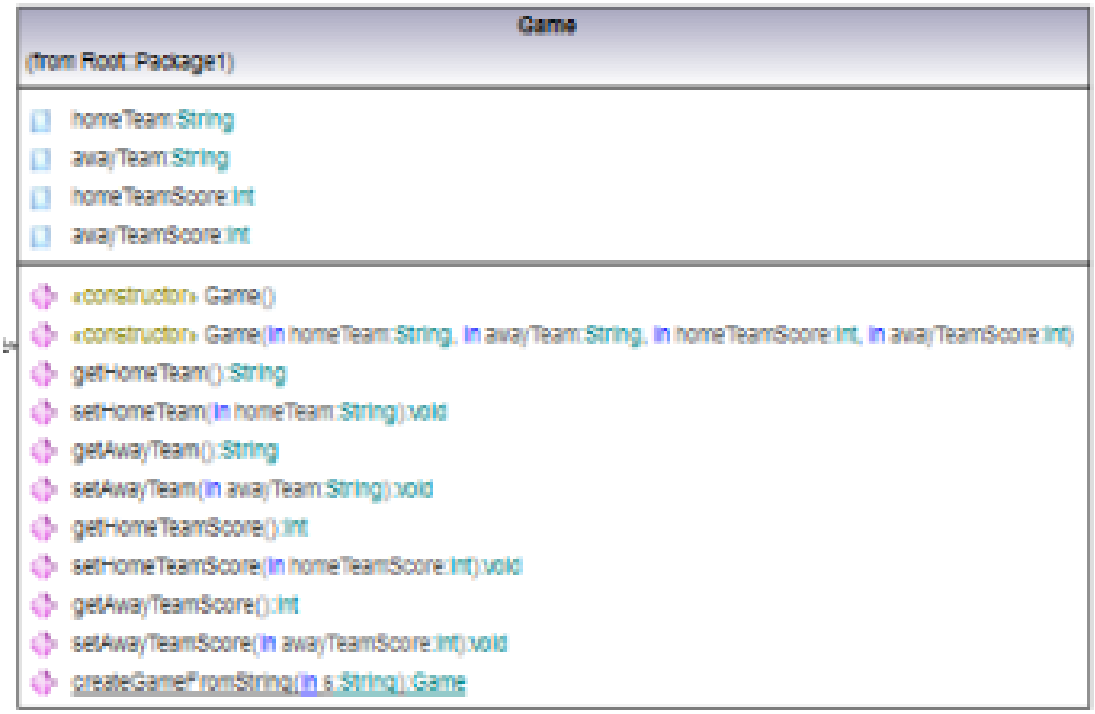
6.1.6 Κλάση HiddenLayer



Η κλάση HiddenLayer, αναπαριστά ένα νευρώνα που βρίσκεται στο κρυφό επίπεδο εισόδου. Περιέχει τον αριθμό του νευρώνα, την τιμή που θα πάρει, καθώς και δύο λίστες, μια για τα βάρη και μια για τα προηγούμενα βάρη που είχε. Ακόμη, περιέχει τον αριθμό του επιπέδου στο οποίο βρίσκεται (αν είναι στο πρώτο ή στο δεύτερο) καθώς και τιμή του λάθους.

Οι μέθοδοι που το αποτελούν είναι ο κατασκευαστής και οι δύο που δίνουν τις τιμές στα βάρη και τα προηγούμενα βάρη.

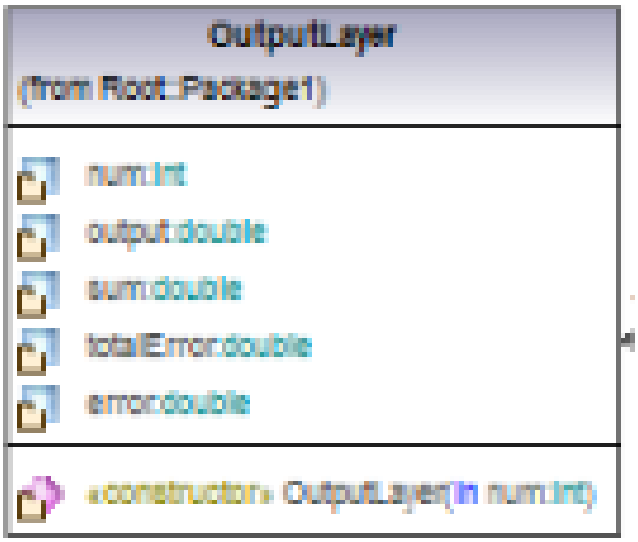
6.1.7 Κλάση Game



Η κλάση Game, αναπαριστά ένα παιχνίδι μεταξύ δύο ομάδων. Τα χαρακτηριστικά της κλάσης είναι το όνομα της εντός έδρας ομάδας, το όνομα της εκτός έδρας ομάδας, ο αριθμός των πόντων που πέτυχε η γηπεδούχος και ο αριθμός πόντων που πέτυχε η φιλοξενούμενη.

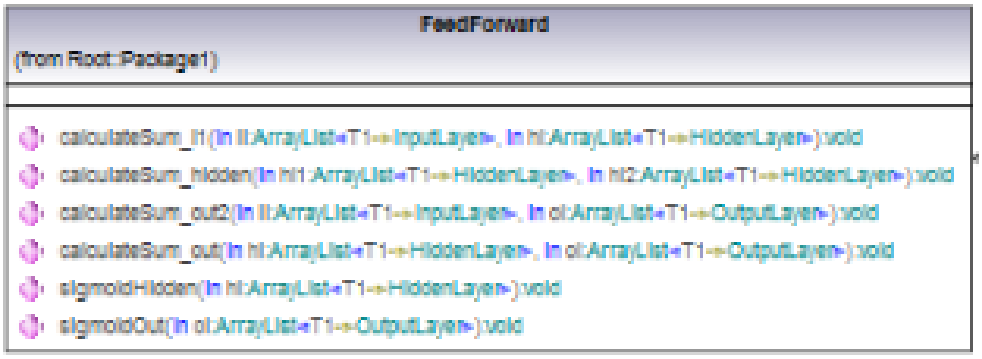
Υπάρχει η μέθοδος του κατασκευαστή, καθώς και ξεχωριστές μέθοδοι για να ορίσουμε και να πάρουμε το κάθε χαρακτηριστικό. Τέλος, η μέθοδος createGameFromString, παίρνει ένα String της μορφής ATLANTA-DALLAS 110-104 για παράδειγμα και το μετατρέπει σε τύπο παιχνίδι.

6.1.8 Κλάση OutputLayer



Η κλάση OutputLayer, αναπαριστά ένα νευρώνα στο επίπεδο εξόδου. Περιέχει σαν χαρακτηριστικά τον αριθμό του νευρώνα, την τιμή εξόδου, το άθροισμα που έχει παραχθεί από τους διάφορους υπολογισμούς του αλγόριθμου και την τιμή του λάθους. Σαν μέθοδο περιέχει τον κατασκευαστή της.

6.1.9 Κλάση FeedForward



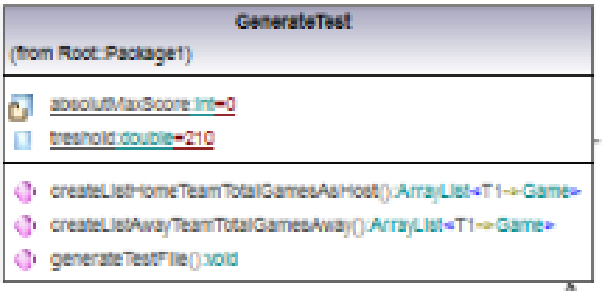
Η κλάση FeedForward είναι αυτή που κάνει την εμπρός τροφοδότηση, κάτι που είναι απαραίτητο συστατικό του αλγόριθμο Back-Propagation. Περιέχει μεθόδους οι οποίες

- Υπολογίζουν το άθροισμα όλων των νευρώνων του πρώτου κρυφού επιπέδου.
- Υπολογίζουν το άθροισμα όλων των νευρώνων του δεύτερου κρυφού επιπέδου
- Υπολογίζουν το άθροισμα όλων των νευρώνων στο επίπεδο εξόδου.
- Υπολογίζουν το αποτέλεσμα της σιγμοειδούς συνάρτησης.

6.1.10 Κλάση BackPropagation

Αυτή η κλάση περιέχει μεθόδους που είναι απαραίτητες για την εκτέλεση του αλγόριθμου. Μεθόδους που υπολογίζουν το λάθος στα διάφορα επίπεδα, καθώς και τις αλλαγές στα βάρη.

6.1.11 Κλάση GenerateTest



Στην κλάση GenerateTest, διαβάζονται οι πληροφορίες για τις ομάδες τις οποίες θέλουμε να προβλέψουμε την έκβαση του αποτελέσματος και μέσω αυτών των πληροφοριών δημιουργούνται τα αρχεία που περιέχουν τα δεδομένα εκπαίδευσης και τα δεδομένα ελέγχου.

Περιέχει μεθόδους που δημιουργούν λίστα με τους αγώνες που έδωσε η εντός έδρας ομάδα στην έδρα της, η φιλοξενούμενη ομάδα εκτός έδρας.

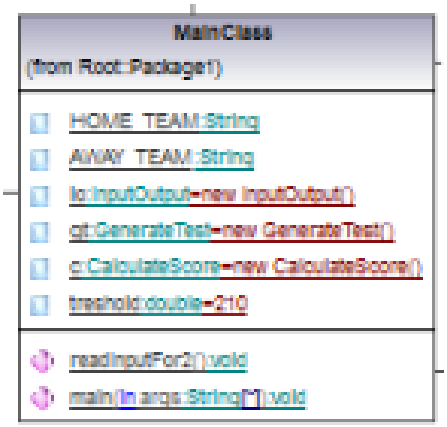
Τα αρχεία εκπαίδευσης και ελέγχου, προκύπτουν από των υπολογισμό των inputs που θέλουμε να δώσουμε στον αλγόριθμο. Αυτά τα Inputs υπολογίζονται βάση των δύο πιο πάνω λιστών.

6.1.12 Κλάση CalculateScore

Η κλάση CalculateScore, είναι η κλάση στην οποία γίνονται οι κύριοι υπολογισμοί του αλγόριθμου Back-Propagation. Περιέχει μεθόδους για την εκπαίδευση, για την

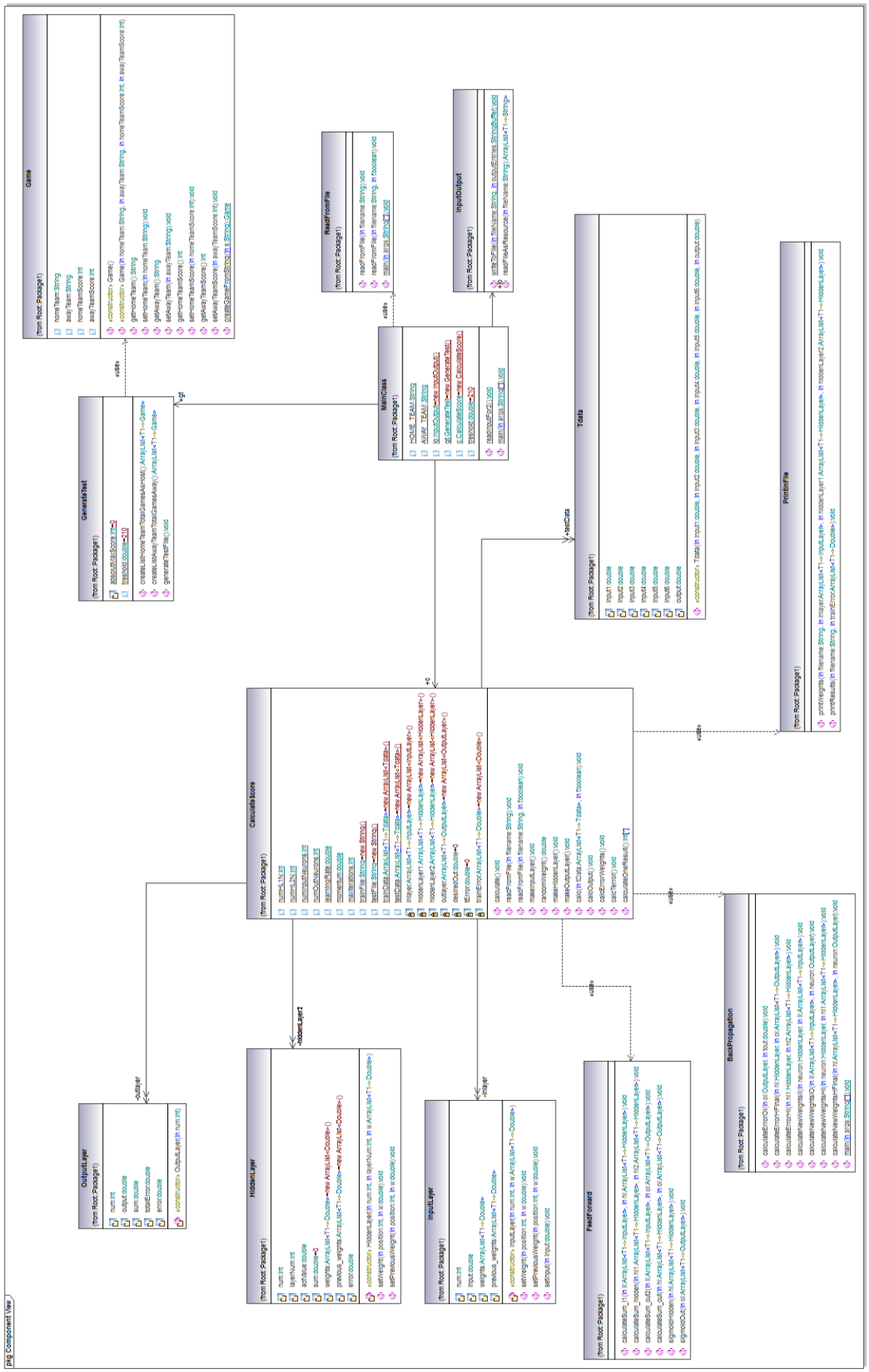
κατασκευή του επιπέδου εισόδου, των κρυφών επιπέδων και του επιπέδου εξόδου, μεθόδους για αρχικοποίηση των βαρών. Υπολογίζονται ακόμη οι εξόδοι και τα λάθη.

6.1.13 Κλάση MainClass



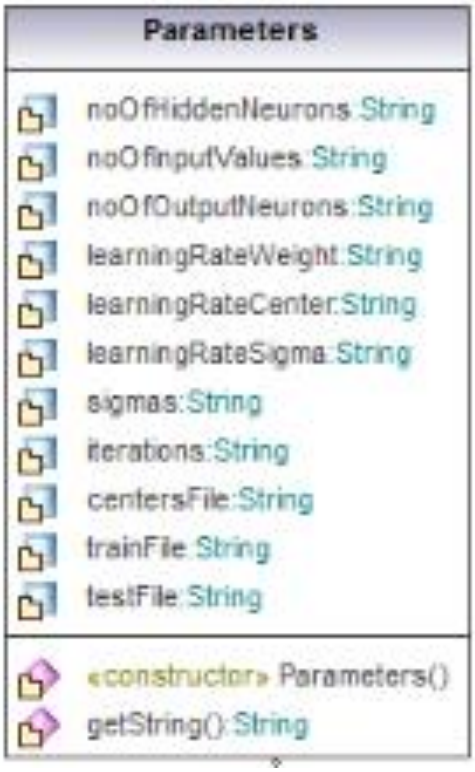
Στην κλάση MainClass περιέχεται η μέθοδος main από την οποία τρέχει το πρόγραμμα. Ο χρήστης μπορεί να επιλέξει για να προβλέψει το αποτέλεσμα ενός αγώνα μεταξύ δύο ομάδων ή να επιλέξει πρόβλεψη για όλους τους συνδυασμούς των ομάδων. Μετά την εκπαίδευση και πρόβλεψη, ο χρήστης ρωτάται για το αν θέλει να συνεχίσει ή όχι.

6.1.14 Διάγραμμα UML για αλγόριθμο Back-Progapation



6.2 Σχεδιασμός και υλοποίηση αλγόριθμου RBF

6.2.1 Κλάση Parameters



Η κλάση Parameters χρησιμοποιείται για να γίνει το διάβασμα του αρχείου που περιέχει τις παραμέτρους που δίνονται στον αλγόριθμο RBF.

Παράδειγμα παραμέτρων:

numHiddenLayerNeurons 2

numInputNeurons 6

numOutputNeurons 1

learningRateWeight 0.5

learningRateCenter 0.5

learningRateSigma 0.5

sigmas 0.2

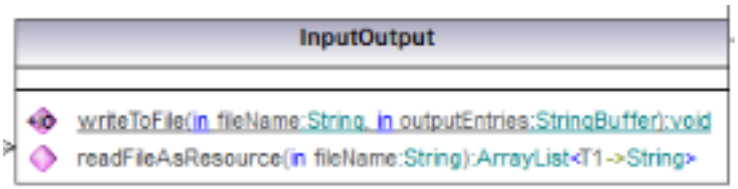
maxIterations 50

centersFile centersFile.txt

trainFile training.txt

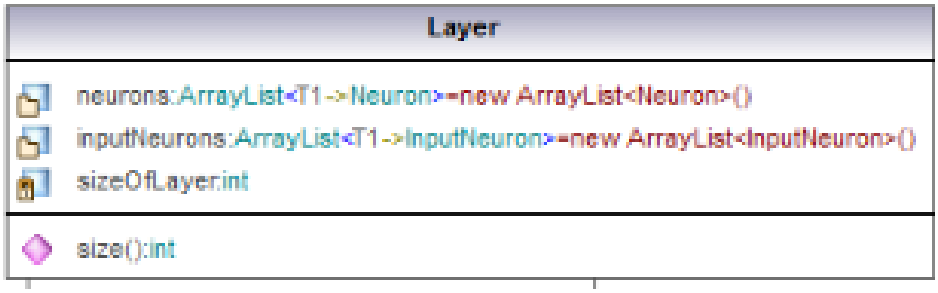
testFile test.txt

6.2.2 Κλάση InputOutput



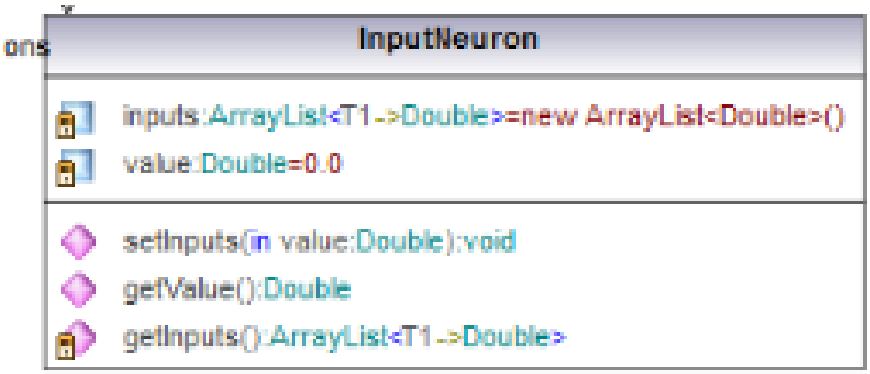
Η κλάση InputOutput χρησιμοποιείται για να διαβαστούν οι πληροφορίες των ομάδων και να γραφεί το αρχείο με τα δεδομένα εκπαίδευσης, ελέχθου, καθώς και το αρχείο με τα κέντρα. Περιέχει δύο μεθόδους, μια για το διάβασμα των αρχείων και μια για την εγγραφή των αρχείων. Χρησιμοποιείται στην κλάση GenerateTest.

6.2.3 Κλάση Layer



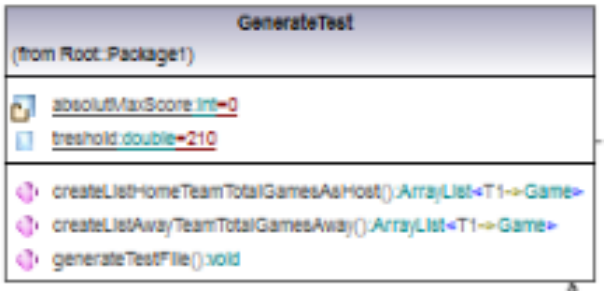
Η κλάση Layer αναπαριστά ένα επίπεδο (εισόδου ή το κρυφό επίπεδο) περιέχοντας τους κόμβους του και το μέγεθος, δηλαδή τον αριθμό των κόμβων.

6.2.4 Κλάση InputNeuron



Η κλάση InputNeuron, περιέχει τους νευρώνες του επιπέδου εισόδου.

6.2.5 Κλάση GenerateTest

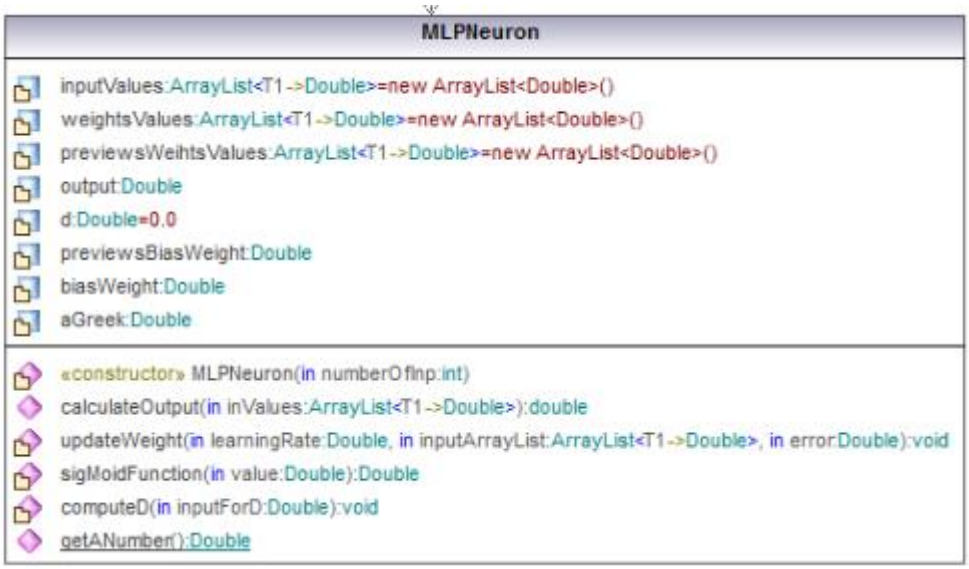


Στην κλάση GenerateTest, διαβάζονται οι πληροφορίες για τις ομάδες τις οποίες θέλουμε να προβλέψουμε την έκβαση του αποτελέσματος και μέσω αυτών των πληροφοριών δημιουργούνται τα αρχεία που περιέχουν τα δεδομένα εκπαίδευσης, τα δεδομένα ελέγχου καθώς επίσης και τα κέντρα.

Περιέχει μεθόδους που δημιουργούν λίστα με τους αγώνες που έδωσε η εντός έδρας ομάδα στην έδρα της, η φιλοξενούμενη ομάδα εκτός έδρας.

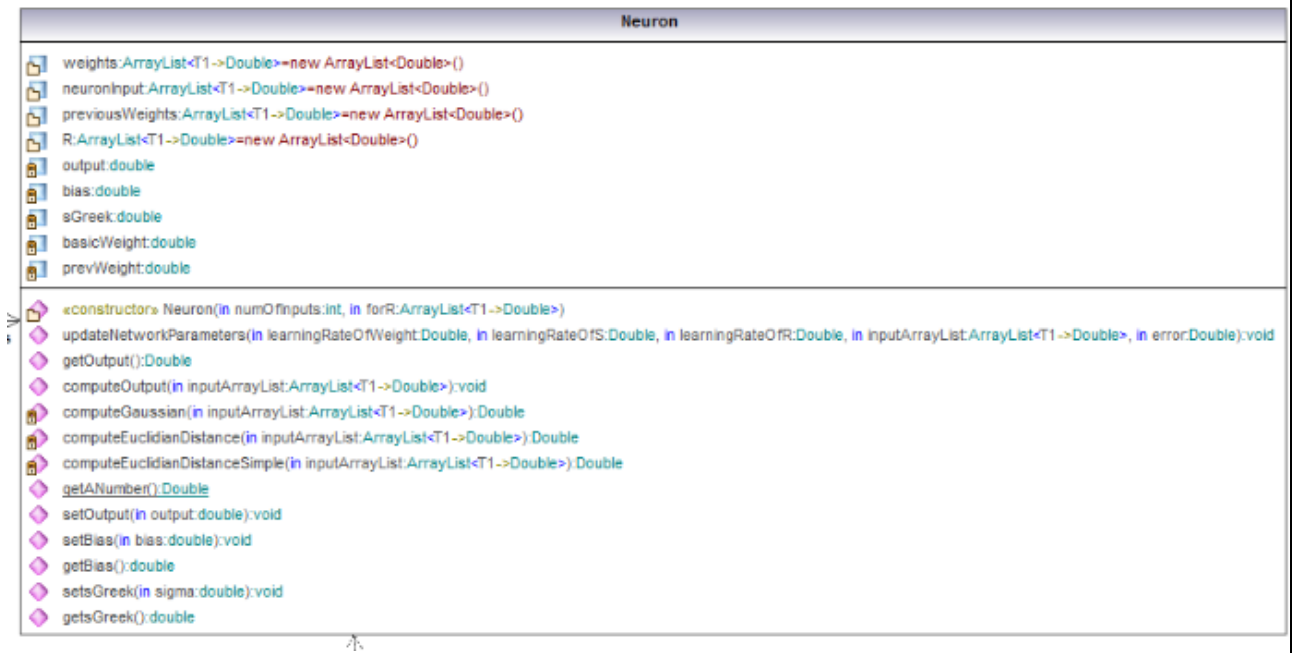
Τα αρχεία εκπαίδευσης και ελέγχου, προκύπτουν από των υπολογισμό των inputs που θέλουμε να δώσουμε στον αλγόριθμο. Αυτά τα Inputs υπολογίζονται βάση των δύο πιο πάνω λιστών.

6.2.6 Κλάση OutputNeuron



Η κλάση OutputNeuron, αντιπροσωπεύει ένα νευρώνα στο επίπεδο εξόδου και περιέχει παρόμοιες μεθόδους με αυτές που χρησιμοποιούνται στα Multilayer Perceptrons.

6.2.7 Κλάση Neuron



Η κλάση Neuron αντιπροσωπεύει ένα νευρώνα του κρυφού επιπέδου. Εδώ γίνονται όλες οι λειτουργίες του αλγόριθμου RBF όπως ο υπολογισμός της ευκλείδιας

απόστασης, η οποία στη συνέχεια περνά από τη γκαουσιανή συνάρτηση, ο υπολογισμός της εξόδου του κάθε νευρώνα και η ανανέωση των παραμέτρων του δικτύου.

6.2.8 Κλάση Game

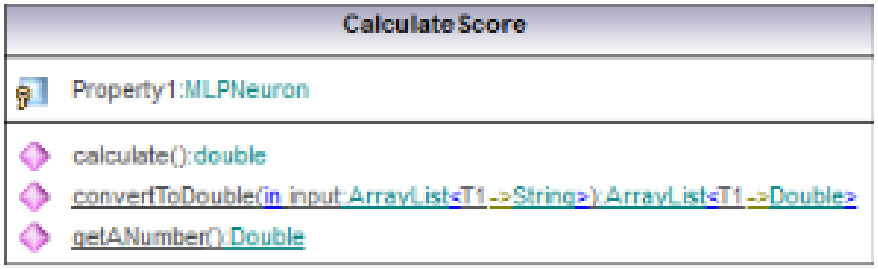
Η κλάση Game, αναπαριστά ένα παιχνίδι μεταξύ δύο ομάδων. Τα χαρακτηριστικά της κλάσης είναι το όνομα της εντός έδρας ομάδας, το όνομα της εκτός έδρας ομάδας, ο αριθμός των πόντων που πέτυχε η γηπεδούχος και ο αριθμός πόντων που πέτυχε η φιλοξενούμενη.

Υπάρχει η μέθοδος του κατασκευαστή, καθώς και ξεχωριστές μέθοδοι για να ορίσουμε και να πάρουμε το κάθε χαρακτηριστικό. Τέλος, η μέθοδος createGameFromString, παίρνει ένα String της μορφής ATLANTA-DALLAS 110-104 για παράδειγμα και το μετατρέπει σε τύπο παιχνίδι.

6.2.9 Κλάση TestTraining

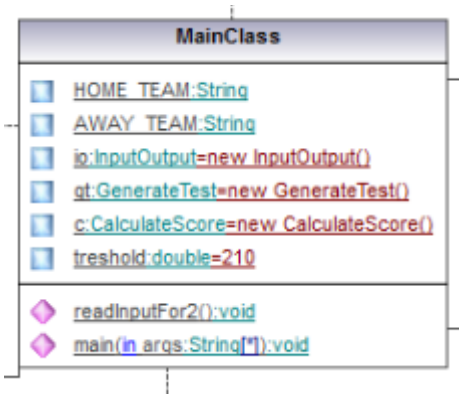
Η κλάση TestTraining χρησιμοποιείται για να διαβάσουμε αρχεία που περιέχουν τα δεδομένα εκπαίδευσης και τα δεδομένα ελέγχου. Περιέχει μέθοδο που διαβάζει το αρχείο και τοποθετεί τα δεδομένα σε ArrayLists.

6.2.10 Κλάση CalculateScore



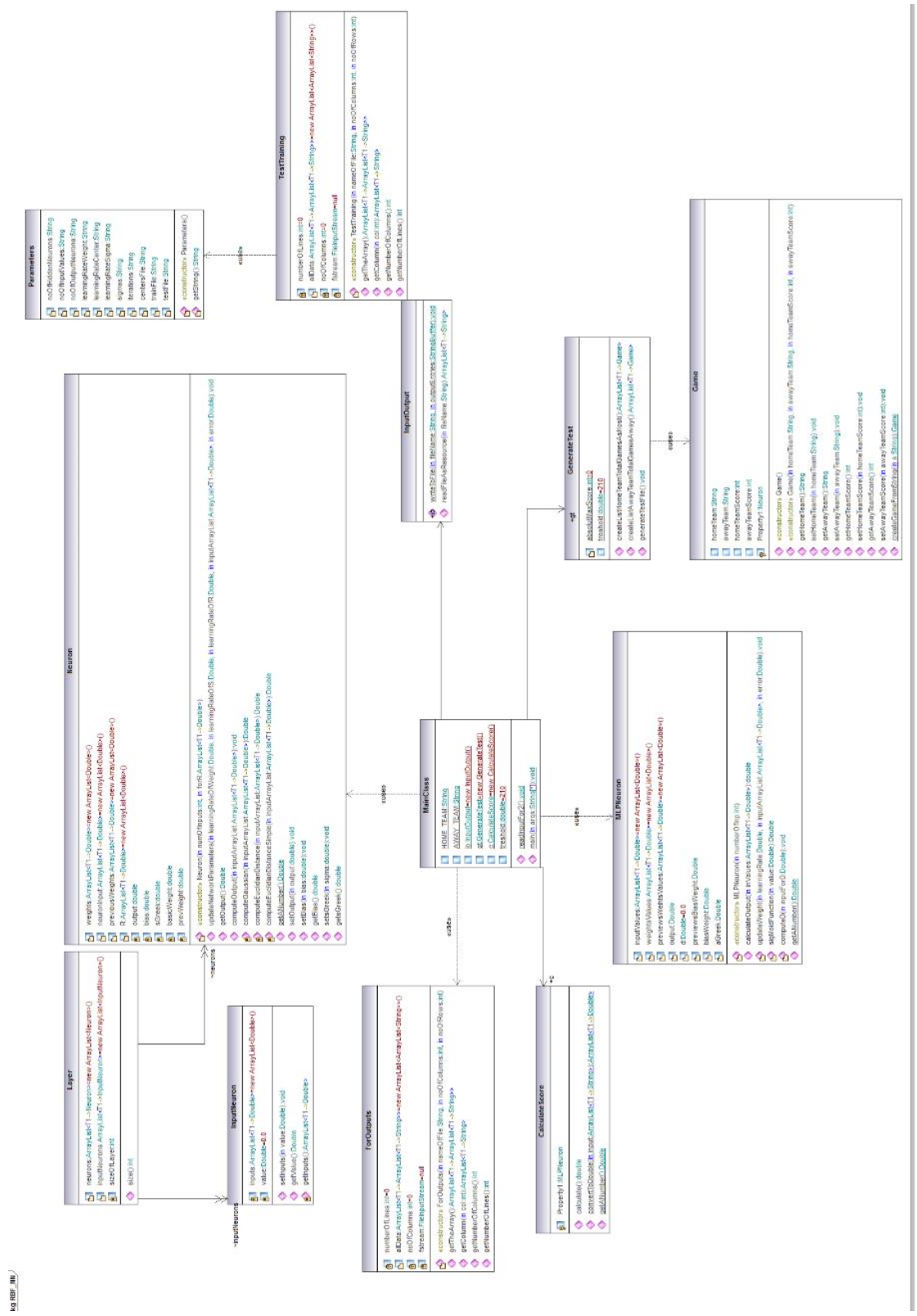
Σε αυτή την κλάση γίνεται η εκπαίδευση του δικτύου, με βάση τα δεδομένα που διαβάζονται από τα αρχεία που περιέχουν τα δεδομένα εκπαίδευσης και τα κέντρα. Σε αυτή την κλάση υπολογίζεται και η έξοδος.

6.2.11 Κλάση MainClass



Στην κλάση MainClass περιέχεται η μέθοδος main από την οποία τρέχει το πρόγραμμα. Ο χρήστης μπορεί να επιλέξει για να προβλέψει το αποτέλεσμα ενός αγώνα μεταξύ δύο ομάδων ή να επιλέξει πρόβλεψη για όλους τους συνδυασμούς των ομάδων. Μετά την εκπαίδευση και πρόβλεψη, ο χρήστης ρωτάται για το αν θέλει να συνεχίσει ή όχι.

6.2.12 Διάγραμμα UML για αλγόριθμο RBF



Κεφάλαιο 7

Αποτελέσματα και συζήτηση

7.1 Αποτελέσματα Αλγόριθμου Back-Propagation

Σε αυτό το σημείο θα αναλύσουμε τα αποτελέσματα που προέκυψαν από τις εκτελέσεις του αλγόριθμου Back-Propagation, τα οποία κάθε φορά ήταν διαφορετικά, ανάλογα με τον αριθμό των παραμέτρων που δίναμε στο δίκτυο. Έγιναν πολλές δοκιμές, με διάφορα δεδομένα και αριθμούς και θα παραθέσουμε μερικά από αυτά, αυτά που κρίναμε ότι ήταν τα πιο σημαντικά.

Σε όλες τις περιπτώσεις δοκιμών, τα δεδομένα ήταν του ίδιου τύπου, τα αποτελέσματα δηλαδή των παλαιότερων αγώνων NBA από το 2004 μέχρι σήμερα. Σε κάποιες εκτελέσεις χρησιμοποιήθηκαν όλα, ενώ σε κάποιες άλλες, επιλέξαμε να χρησιμοποιήσουμε για λιγότερες season, για να δούμε αν έπαιζε ρόλο η μεγάλη διαφορά του χρόνου. Όσον αφορά τα δεδομένα ελέγχου αυτά είναι η αγωνιστική περίοδος 2012/2013.

Στον αλγόριθμο Back-Propagation, επιχειρήσαμε να αυξήσουμε ή να μειώσουμε τους νευρώνες στο πρώτο κρυφό επίπεδο, να δοκιμάσουμε να προσθέσουμε δεύτερο επίπεδο καθώς επίσης να αλλάξουμε τις εισόδους που δίνουμε.

Ακολουθούν παραδείγματα που δείχνουν τις διαδικασίες εκπαίδευσης.

Πρώτη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 6

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:6

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 6

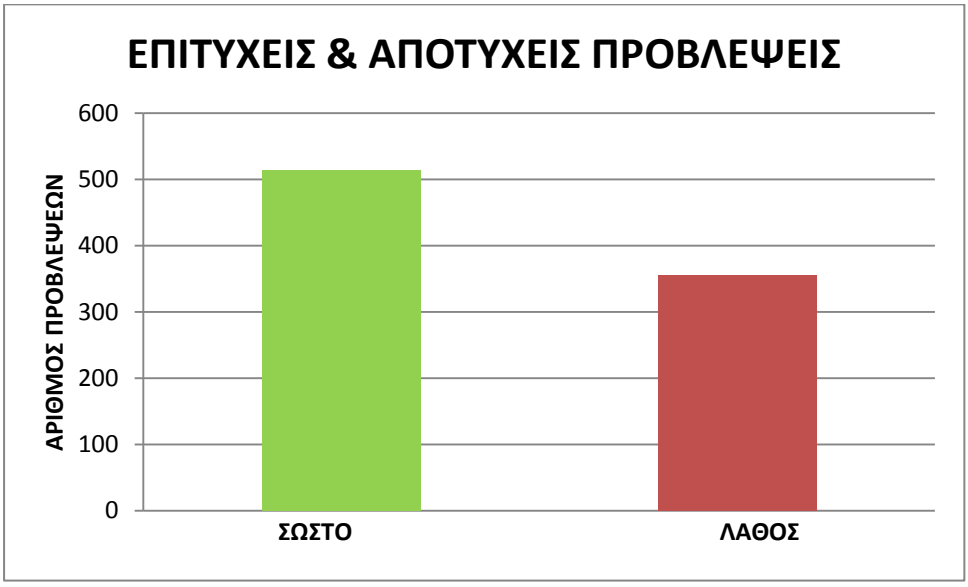
Ρυθμός Μάθησης: 0.6

Ορμή: 0.5

Αριθμός Επαναλήψεων: 100

Περιόδοι Εκπαίδευσης: 2007-2013

Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.59

Στην πρώτη εκτέλεση, επιλέξαμε να χρησιμοποιήσουμε 6 νευρώνες εισόδου, τους πρώτους 6 που αναφέραμε στο διάνυσμα για τα δεδομένα εισόδου. Χρησιμοποιήσαμε 2 κρυφά επίπεδα, που το καθένα περιείχε 6 νευρώνες καθώς και ένα νευρώνα εξόδου. Ο ρυθμός μάθησης επιλέξαμε να είναι 0.6 ενώ ο όρος ορμής 0.5. Ορίσαμε το κατώφλι στους 200 πόντους. Για τον υπολογισμό των εξόδων, λάβαμε υπόψη τις χρονιές 2007 μέχρι 2013 και οι προβλέψεις αφορούν την τελευταία χρονιά. Παρατηρούμε ότι έχουμε ποσοστό επιτυχών προβλέψεων 59%. Τα αποτελέσματα μπορούν να κριθούν ικανοποιητικά ωστόσο όπως θα δούμε σε επόμενες εκτελέσεις, υπάρχει βελτίωση.

Δεύτερη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 6

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:6

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 6

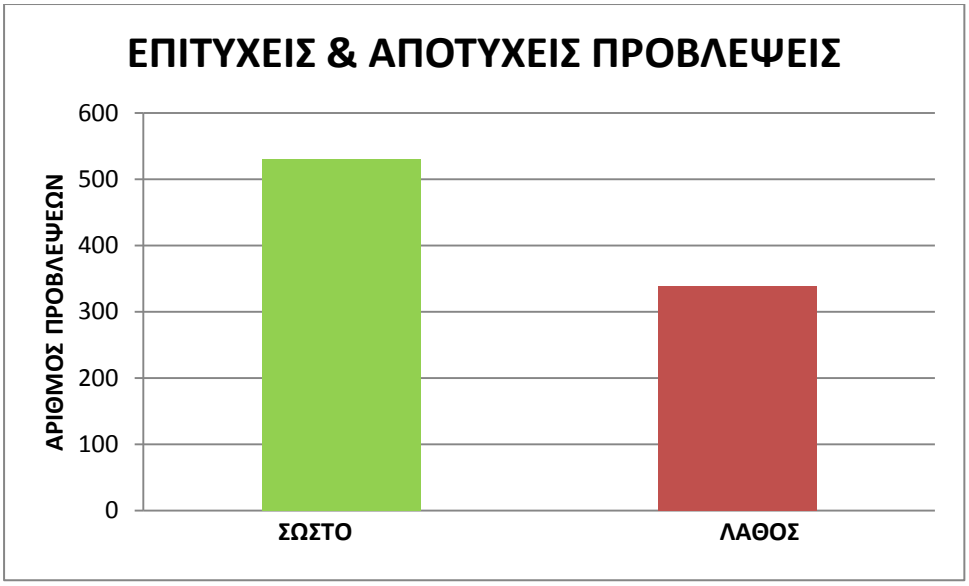
Ρυθμός Μάθησης: 0.7

Ορμή: 0.5

Αριθμός Επαναλήψεων: 1000

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.61

Στην δεύτερη εκτέλεση, επιλέξαμε να χρησιμοποιήσουμε 6 νευρώνες εισόδου όπως και στην πρώτη εκτέλεση. Χρησιμοποιήσαμε 2 κρυφά επίπεδα, που το καθένα περιείχε 6 νευρώνες καθώς και ένα νευρώνα εξόδου. Ο ρυθμός μάθησης επιλέξαμε να είναι 0.7 ενώ ο όρος ορμής 0.5. Ορίσαμε το κατώφλι στους 200 πόντους. Για τον

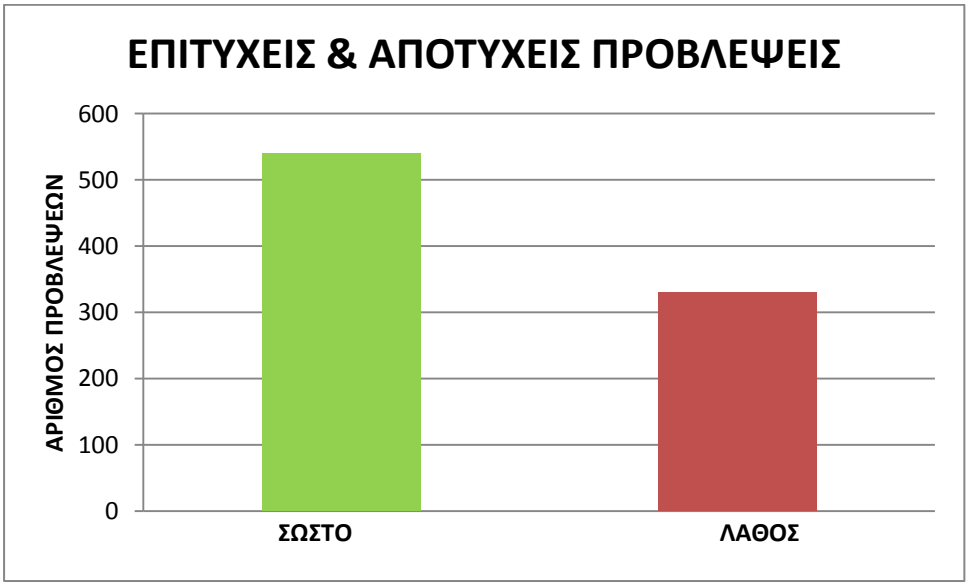
υπολογισμό των εξόδων, λάβαμε υπόψη τις χρονιές 2004 μέχρι 2013 και οι προβλέψεις αφορούν την τελευταία χρονιά. Παρατηρούμε ότι έχουμε ποσοστό επιτυχών προβλέψεων 61%. Σε αντίθεση με την πρώτη εκτέλεση, αυξήσαμε τις επαναλήψεις σε 1000 από 100 και πήραμε ελάχιστα καλύτερα αποτελέσματα.

Παρατηρούμε πιο κάτω ότι το λάθος εκπαίδευσης μειώνεται ακόμα περισσότερο στις περισσότερες επαναλήψεις.

Τρίτη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 6
Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:3
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0
Ρυθμός Μάθησης: 0.7
Ορμή: 0.5
Αριθμός Επαναλήψεων: 100
Περιόδοι Εκπαίδευσης: 2004-2013
Κατώφλι: 200 Πόντοι.



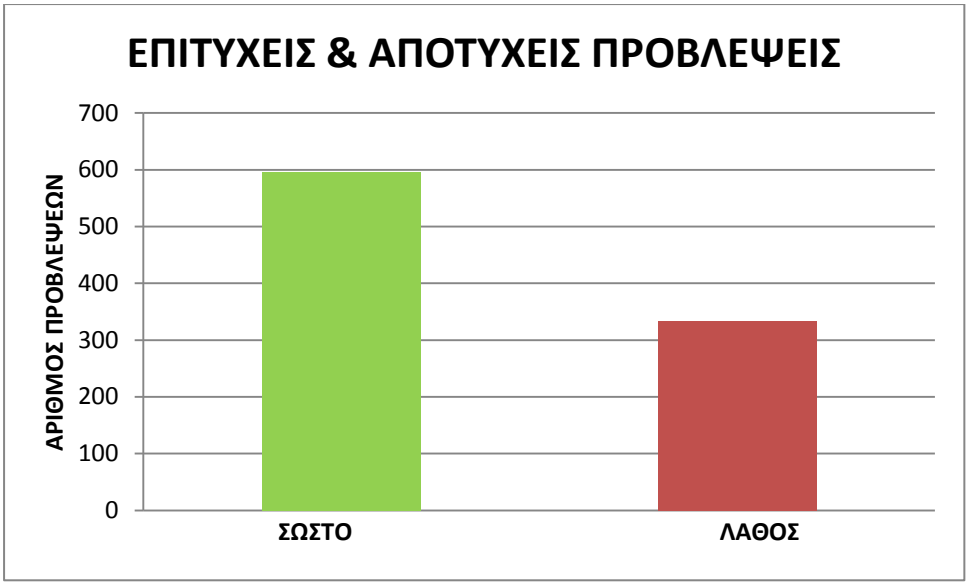
Ποσοστό επιτυχίας: 0.62

Στην τρίτη εκτέλεση, επιλέξαμε να χρησιμοποιήσουμε 6 νευρώνες εισόδου όπως και στην πρώτη εκτέλεση. Όμως αφαιρέσαμε το δεύτερο επίπεδο και δοκιμάσαμε να δούμε τα αποτελέσματα με 3 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης επιλέξαμε να είναι 0.7 ενώ ο όρος ορμής 0.5, όπως και στις προηγούμενες εκτελέσεις. Ορίσαμε το κατώφλι στους 200 πόντους όπως και πριν. Για τον υπολογισμό των εξόδων, λάβαμε υπόψη τις χρονιές 2004 μέχρι 2013 και οι προβλέψεις αφορούν την τελευταία χρονιά. Παρατηρούμε ότι έχουμε ποσοστό επιτυχών προβλέψεων 62%. Ελαφρώς καλύτερα αποτελέσματα χωρίς το δεύτερο κρυφό επίπεδο και γρηγορότερη εκτέλεση λόγω της μείωσης της πολυπλοκότητας.

Τέταρτη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 6
Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:12
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0
Ρυθμός Μάθησης: 0.7
Ορμή: 0.5
Αριθμός Επαναλήψεων: 100
Περιόδοι Εκπαίδευσης: 2004-2013
Κατώφλι: 200 Πόντοι



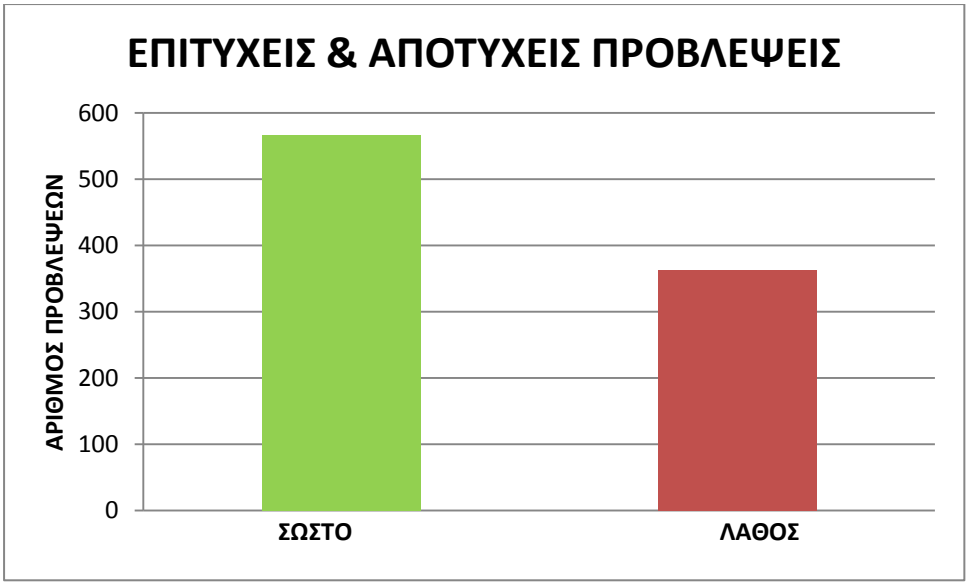
Ποσοστό επιτυχίας: 0.63

Στην τέταρτη εκτέλεση επιλέξαμε να χρησιμοποιήσουμε τα ίδια δεδομένα με την προηγούμενη, με τη διαφορά ότι σε αυτή την περίπτωση χρησιμοποιήσαμε 12 νευρώνες στο πρώτο κρυφό επίπεδο. Τα αποτελέσματα που πήραμε ήταν λίγο καλύτερα και άγγιξαν το 63% όμως ο χρόνος εκτέλεσης αυξήθηκε κατά πολύ σε σχέση με την προηγούμενη εκτέλεση. Παρατηρήσαμε όμως ότι όταν οι νευρώνες στο κρυφό επίπεδο είναι περισσότεροι από αυτούς του επιπέδου εισόδου, το δίκτυο δουλεύει καλύτερα και παράγει καλύτερα αποτελέσματα.

Πέμπτη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 6
Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:6
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 6
Ρυθμός Μάθησης: 0.6
Ορμή: 0.5
Αριθμός Επαναλήψεων: 10
Περιόδοι Εκπαίδευσης: 2008-2013
Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.60

Στην πέμπτη εκτέλεση, επιλέξαμε να χρησιμοποιήσουμε 6 νευρώνες εισόδου όπως και στην πρώτη εκτέλεση. Χρησιμοποιήσαμε 2 κρυφά επίπεδα, που το καθένα περιείχε 6 νευρώνες καθώς και ένα νευρώνα εξόδου. Ο ρυθμός μάθησης επιλέξαμε να είναι 0.7 ενώ ο όρος ορμής 0.5. Ορίσαμε το κατώφλι στους 200 πόντους. Για τον υπολογισμό των εξόδων, λάβαμε υπόψη τις χρονιές 2008 μέχρι 2013 και οι προβλέψεις αφορούν την τελευταία χρονιά. Δοκιμάσαμε λοιπόν να μειώσουμε τις σεζόν για να δούμε αν οι προηγούμενες εκτελέσεις, επηρεάζονταν από το ότι λαμβάναμε υπόψη πολύ παλαιότερα αποτελέσματα. Το ποσοστό που πήραμε δεν ήταν πολύ διαφορετικό από τα προηγούμενα όπου χρησιμοποιήθηκαν όλες οι σεζόν, έτσι δεν μπορούμε να πούμε ότι το δίκτυο επηρεάζεται. Ασφαλώς αν πηγαίναμε πιο πίσω και χρησιμοποιούσαμε 30 σεζόν, ίσως εκεί να υπήρχε διαφορά.
Το ποσοστό επιτυχίας ήταν 60%.

Έκτη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 8

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:6

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 6

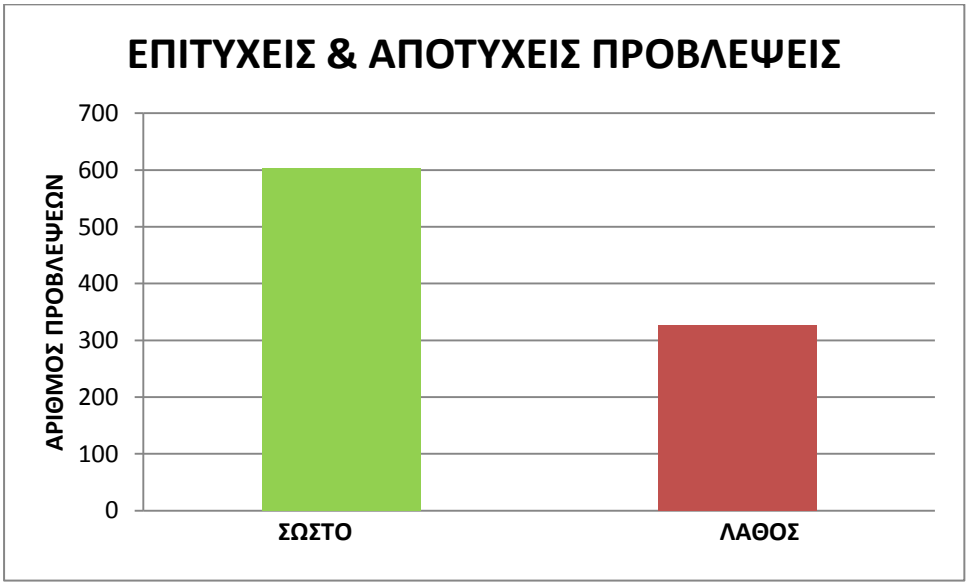
Ρυθμός Μάθησης: 0.6

Ορμή: 0.5

Αριθμός Επαναλήψεων: 10

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.65

Σε αυτή την εκτέλεση επιχειρήσαμε να προσθέσουμε περισσότερες εισόδους.
Χρησιμοποιήσαμε 8 εισόδους, οι οποίες ήταν οι εξής:

Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της

Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της

Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας

Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας

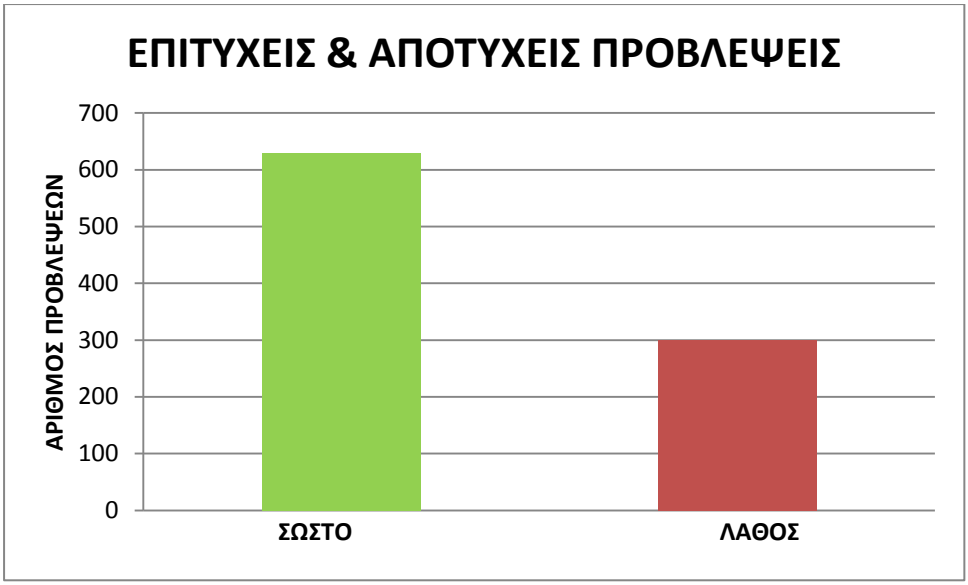
Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες

Διατηρήσαμε το κατώφλι στους 200 πόντους καθώς και τον ρυθμό μάθησης και τον όρο ορμής. Οι περίοδοι εκπαίδευσης ήταν 2004-2013. Τα αποτελέσματα ήταν καλύτερα, αφού πήραμε ποσοστό επιτυχίας 65%.

Έβδομη εκτέλεση

Αριθμός εισόδων: 8

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:16
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0
Ρυθμός Μάθησης: 0.6
Ορμή: 0.5
Αριθμός Επαναλήψεων: 10
Περιόδοι Εκπαίδευσης: 2004-2013
Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.67

Και σε αυτή την εκτέλεση επιχειρήσαμε να προσθέσουμε περισσότερες εισόδους. Χρησιμοποιήσαμε 8 εισόδους, οι οποίες ήταν οι εξής:

Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της
Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της
Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας
Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας
Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες
Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες

Αυξήσαμε τους νευρώνες στο πρώτο κρυφό επίπεδο και αφαιρέσαμε το δεύτερο επίπεδο. Τα αποτελέσματα ήταν καλύτερα αφού πήραμε ποσοστό της τάξης του 67%.

Όπως και σε προηγούμενες περιπτώσεις, η αύξηση των νευρώνων του κρυφού επιπέδου, μας έφερε καλύτερα αποτελέσματα.

Όγδοη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 10

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:10

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0

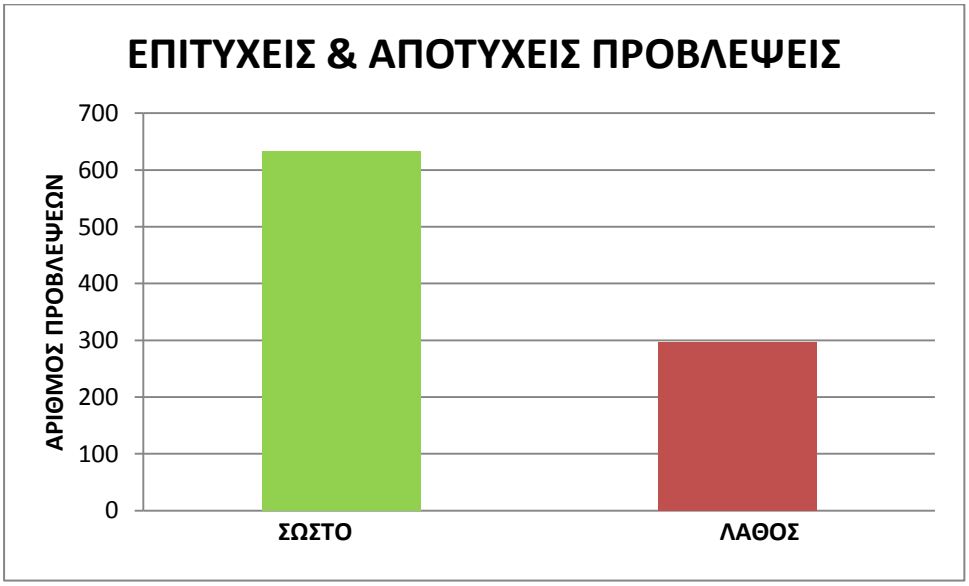
Ρυθμός Μάθησης: 0.6

Ορμή: 0.5

Αριθμός Επαναλήψεων: 100

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.67

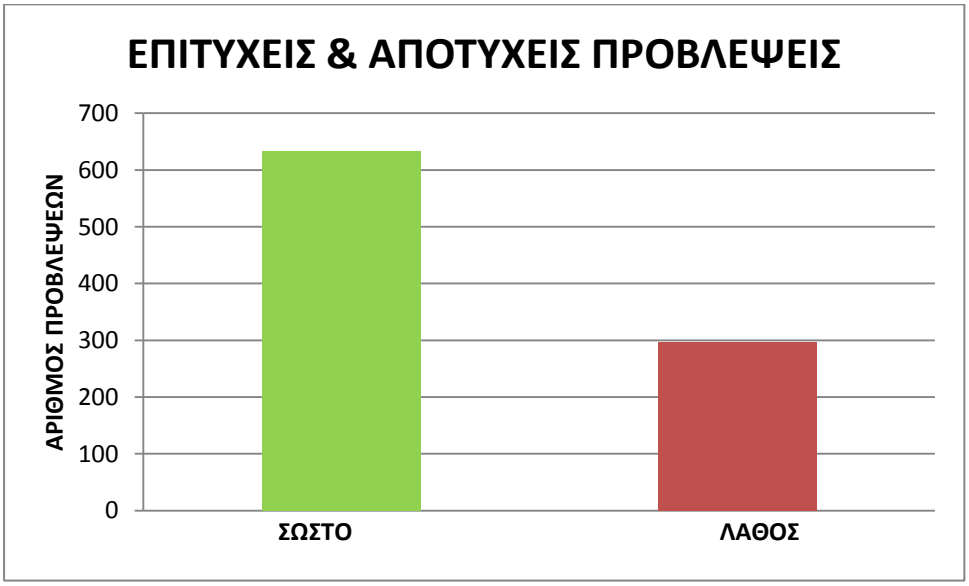
Σε αυτή την εκτέλεση επιχειρήσαμε να προσθέσουμε ακόμα περισσότερες εισόδους. Χρησιμοποιήσαμε 10 εισόδους, τις πρώτες 10 που αναφέρονται στο κεφάλαιο δεδομένα εισόδου. Διατηρήσαμε το κατώφλι στους 200 πόντους καθώς και τον ρυθμό

μάθησης και τον όρο ορμής. Οι περίοδοι εκπαίδευσης παρέμειναν οι ίδιοι. Τα αποτελέσματα ήταν αισθητά καλύτερα, αφού πήραμε ποσοστό επιτυχίας 67%.

Έννατη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 10
Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:10
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0
Ρυθμός Μάθησης: 0.6
Ορμή: 0.5
Αριθμός Επαναλήψεων: 1000
Περίοδοι Εκπαίδευσης: 2004-2013
Κατώφλι: 200 Πόντοι.



Ποσοστό επιτυχίας: 0.68

Σε αυτή την εκτέλεση, κρατήσαμε ακριβώς τις ίδιες παραμέτρους με την προηγούμενη, με μόνη διαφορά ότι αλλάξαμε τον αριθμό επαναλήψεων. Αφού ο υπολογισμούς με 10 εισόδους ήταν ο καλύτερος μέχρι τώρα, προσπαθήσαμε να τον βελτιώσουμε με την αύξηση των επαναλήψεων. Τα αποτελέσματα δεν βελτιώθηκαν

πολύ, και πήραμε ποσοστό επιτυχίας 68%. Αυτό το ποσοστό ήταν και το καλύτερο που πήραμε από τους δύο αλγόριθμους, για περιπτώσεις με κατώφλι τους 200 πόντους.

Δέκατη εκτέλεση:

Δεδομένα:

Αριθμός εισόδων: 10

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:10

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 10

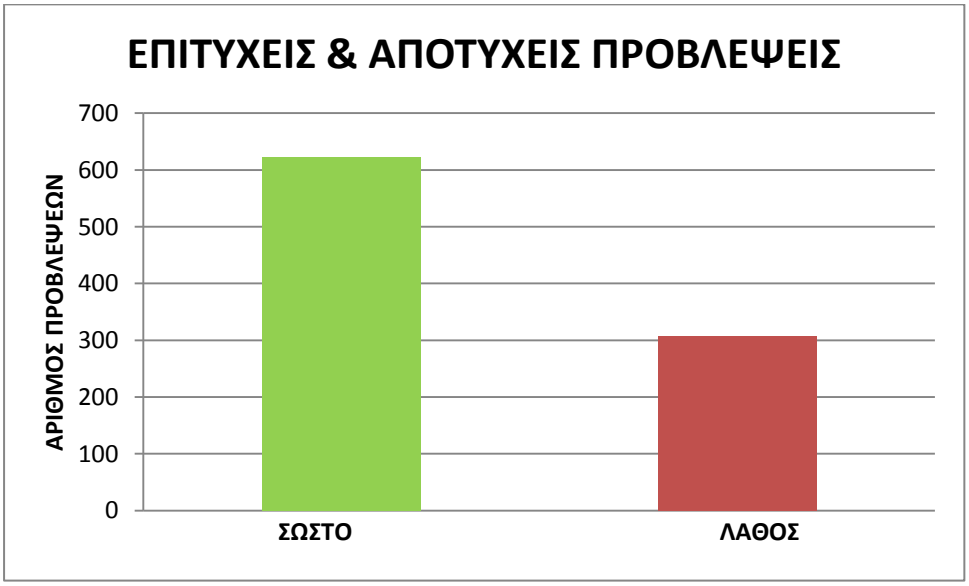
Ρυθμός Μάθησης: 0.6

Ορμή: 0.5

Αριθμός Επαναλήψεων: 1000

Περίοδοι Εκπαίδευσης: 2004-2013

Κατώφλι: 200 Πόντοι.

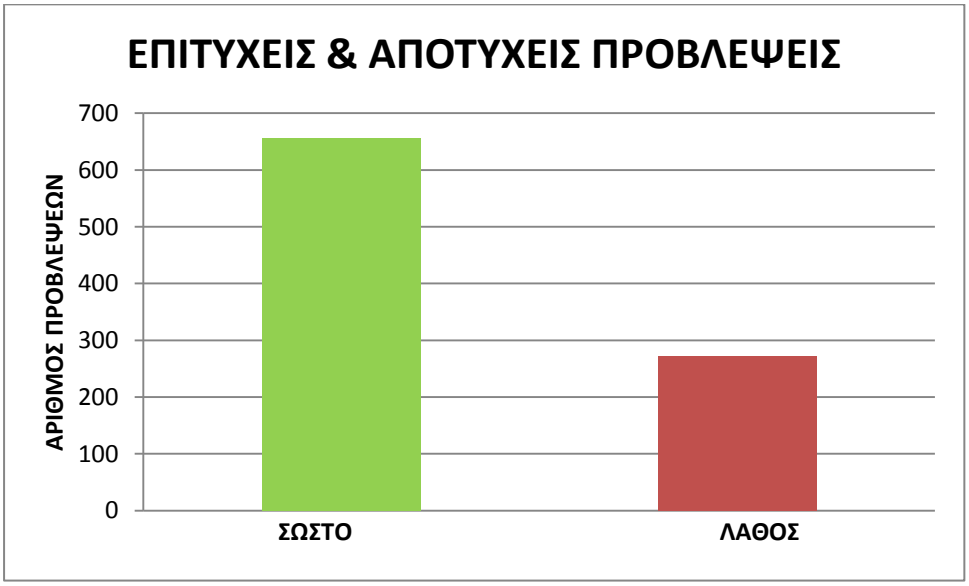


Ποσοστό επιτυχίας: 0.65

Σε αυτή την εκτέλεση, δοκιμάσαμε και πάλι να βελτιώσουμε τα ποσοστά, χρησιμοποιώντας 10 εισόδους, όμως σε αντίθεση με τις δύο προηγούμενες περιπτώσεις, προσθέσαμε και δεύτερο επίπεδο. Τα αποτελέσματα δεν βελτιώθηκαν, ενώ αυξήθηκε και η πολυπλοκότητα του δικτύου.

Ενδέκατη Εκτέλεση:

Αριθμός εισόδων: 10
Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο:20
Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο:
Ρυθμός Μάθησης: 0.6
Ορμή: 0.5
Αριθμός Επαναλήψεων: 1000
Περίοδοι Εκπαίδευσης: 2004-2013
Κατώφλι: 200 Πόντοι.

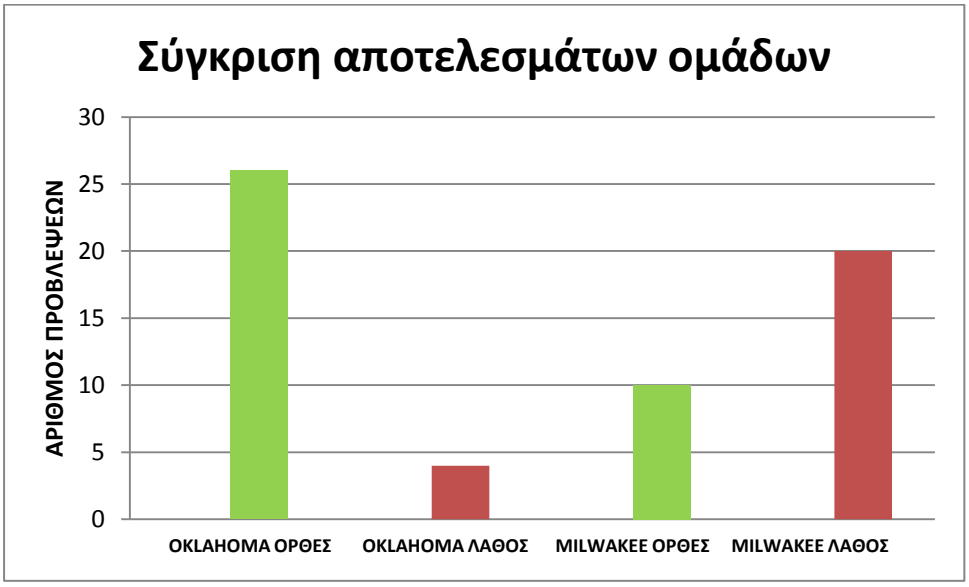


Ποσοστό επιτυχίας: 0.70

Αυτή η εκτέλεση, μας έχει δώσει τα καλύτερα αποτελέσματα. Χρησιμοποιήσαμε 10 εισόδους και 20 νευρώνες στο κρυφό επίπεδο. Δεν χρησιμοποιήσαμε δεύτερο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.6, ενώ ο όρος ορμής 0.5. Ο αριθμός των επαναλήψεων ήταν 1000 και οι περίοδοι εκτέλεσης ήταν από το 2004 μέχρι το 2013.

Το κατώφλι παρέμεινε το ίδιο, στους 200 πόντους. Το ποσοστό επιτυχίας, ανήλθε στο 70%, ποσοστό που αποτελεί και το καλύτερο που έχουμε πετύχει. Αυτό οφείλεται στον μεγάλο αριθμό εισόδων που έχουμε δώσει, καθώς και στην αύξηση των νευρώνων του κρυφού επιπέδου. Ο χρόνος εκτέλεσης είναι μεγαλύτερος σε σχέση με προηγούμενες εκτελέσεις όπου οι νευρώνες στο κρυφό επίπεδο ήταν λιγότεροι, όμως δεν φτάνει σε απαγορευτικό σημείο.

Σύγκριση αποτελεσμάτων μεταξύ δύο ομάδων:



Αριθμός εισόδων: 10

Αριθμός Κρυφών Νευρώνων στο πρώτο κρυφό επίπεδο: 20

Αριθμός Κρυφών Νευρώνων στο δεύτερο κρυφό επίπεδο: 0

Ρυθμός Μάθησης: 0.6

Ορμή: 0.5

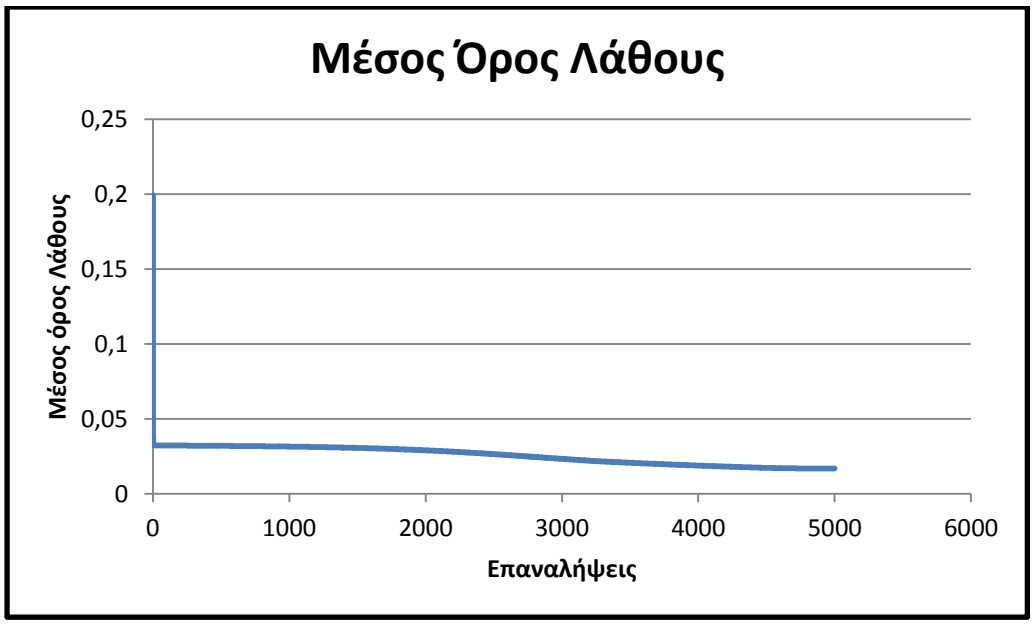
Αριθμός Επαναλήψεων: 100

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι: 200 Πόντοι.

Στο πιο πάνω γράφημα, παρατηρούμε τις ορθές προβλέψεις για τους τελευταίους αγώνες που έδωσαν δύο ομάδες στην έδρα τους με όλες τις αντίστοιχες αντιπάλους. Παρατηρούμε ότι για την ομάδα της OKLAHOMA, ο αλγόριθμος προέβλεψε σωστά 26 από τους 30 αγώνες. Αντίθετα, για την ομάδα MILWAKEE, μόλις 10 από τις 30 προβλέψεις ήταν ορθές. Καταλήξαμε στο συμπέρασμα ότι κάποιες από τις ομάδες, αυτές που παρουσίαζαν ασταθή αποτελέσματα, δυσκόλευαν το δίκτυο στο να προβλέψει σωστά το αποτέλεσμα τους.

Μέσος Όρος Λάθους:



Η γραφική παράσταση του λάθους κατά τις περιόδους εκπαίδευσης του δικτύου, φαίνεται στην εικόνα. Στον άξονα x, βρίσκεται ο αριθμός των επαναλήψεων που έτρεξε το δίκτυο κατά την εκπαίδευσή του. Στον άξονα y παρουσιάζεται ο μέσος όρος λάθους. Η γραφική παράσταση αντιπροσωπεύει όλες τις πιο πάνω εκτελέσεις με κάποιες μικρές διαφορές στις τιμές του λάθους. Η τάση να μειώνεται το λάθος εκπαίδευσης παρατηρείται σε όλες τις περιπτώσεις αφού δεν παρουσιάζονται οποιαδήποτε προβλήματα. Φαίνεται καθαρά ότι το λάθος εκπαίδευσης όσο οι επαναλήψεις αυξάνονται, μειώνεται. Έχουμε φτάσει σε ένα ικανοποιητικό αριθμό λάθους στην εκτέλεση που μας έδωσε τα καλύτερα αποτελέσματα.

7.2 Αποτελέσματα Αλγόριθμου RBF

Σε αυτό το σημείο θα αναλύσουμε τα αποτελέσματα που προέκυψαν από τις εκτελέσεις του αλγόριθμου RBF. Όπως και στον αλγόριθμο Back-Propagation, κάθε φορά ήταν διαφορετικά, ανάλογα με τον αριθμό των παραμέτρων που δίναμε στο δίκτυο. Έγιναν πολλές δοκιμές, με διάφορα δεδομένα και αριθμούς και θα παραθέσουμε μερικά από αυτά, αυτά που κρίναμε ότι ήταν τα πιο σημαντικά.

Σε όλες τις περιπτώσεις δοκιμών, τα δεδομένα ήταν του ίδιου τύπου, τα αποτελέσματα δηλαδή των παλαιότερων αγώνων NBA από το 2004 μέχρι σήμερα. Σε κάποιες εκτελέσεις χρησιμοποιήθηκαν όλα, ενώ σε κάποιες άλλες, επιλέξαμε να χρησιμοποιήσουμε για λιγότερες season, για να δούμε αν έπαιζε ρόλο η μεγάλη διαφορά του χρόνου. Όσον αφορά τα δεδομένα ελέγχου αυτά είναι η αγωνιστική περίοδος 2012/2013.

Πρώτη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 6

Νευρώνες Κρυφού Επιπέδου: 3

Νευρώνες Εξόδου: 1

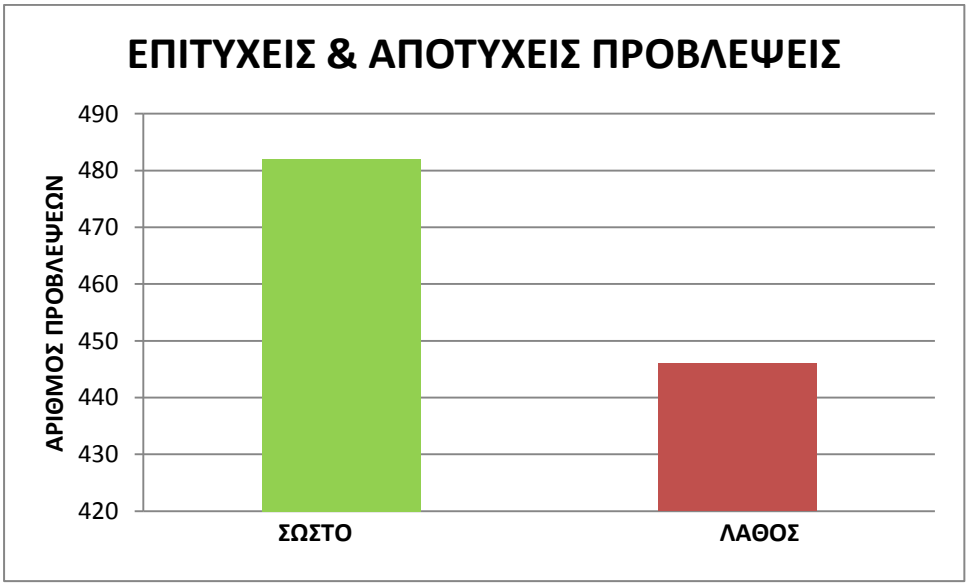
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.51

Στην πρώτη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 6 εισόδους, τις πρώτες 6 που αναφέρονται στα δεδομένα εκπαίδευσης, και 3 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 1000. Τα αποτελέσματα που πήραμε δεν ήταν και τα καλύτερα δυνατά και ήταν της τάξης του 51%.

Δεύτερη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 6

Νευρώνες Κρυφού Επιπέδου: 6

Νευρώνες Εξόδου: 1

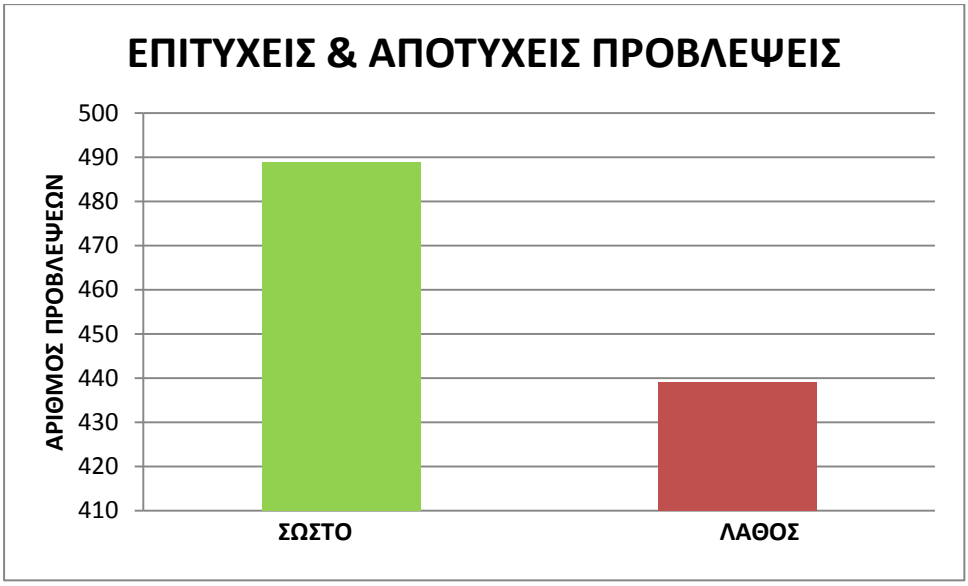
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.53

Στην δεύτερη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 6 εισόδους, τις πρώτες 6 που αναφέρονται στα δεδομένα εκπαίδευσης, και 6 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 1000. Ουσιαστικά η μόνη διαφορά από την πρώτη εκτέλεση ήταν η αλλαγή στον αριθμό των νευρώνων του κρυφού επιπέδου

Παρατηρήθηκε μικρή αύξηση στο ποσοστό επιτυχίας που ανήλθε στο 53%.

Τρίτη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 6

Νευρώνες Κρυφού Επιπέδου: 12

Νευρώνες Εξόδου: 1

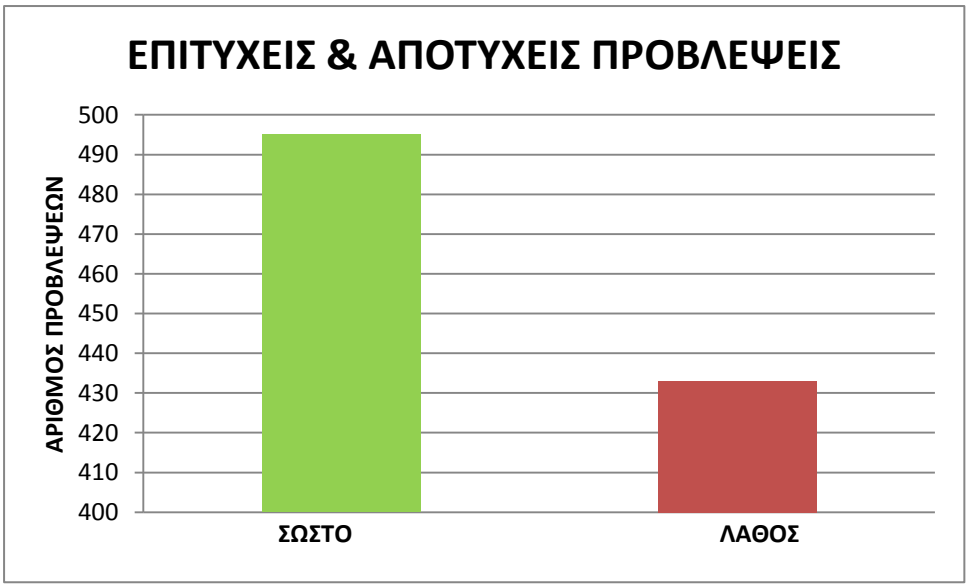
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.54

Στην τρίτη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 6 εισόδους, τις πρώτες 6 που αναφέρονται στα δεδομένα εκπαίδευσης, και 12 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 100. Αυξήσαμε και πάλι τους νευρώνες στο κρυφό επίπεδο και παρατηρήθηκε μικρή αύξηση στο ποσοστό επιτυχίας που άγγιξε το 54%

Τέταρτη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 8

Νευρώνες Κρυφού Επιπέδου: 4

Νευρώνες Εξόδου: 1

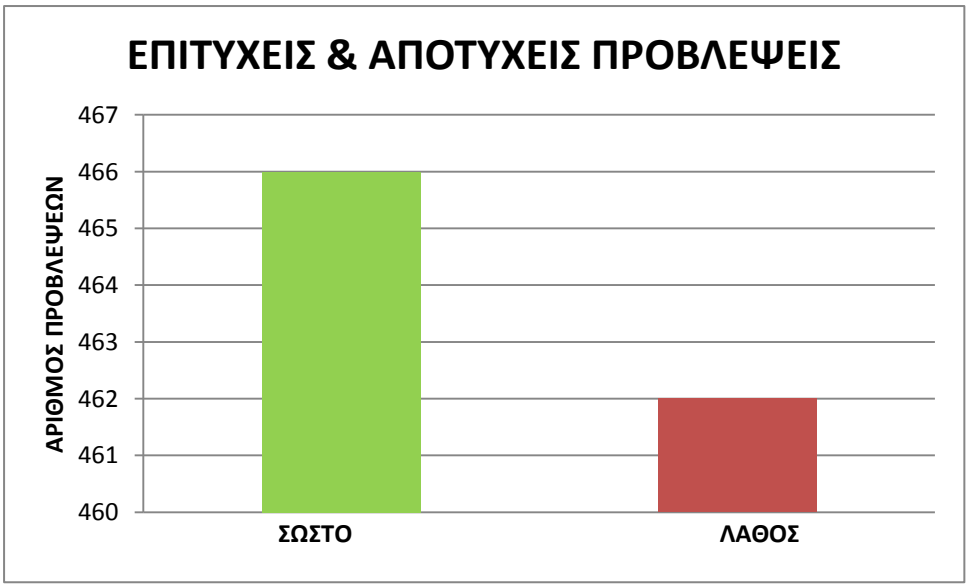
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.50

Στην τέταρτη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 8 εισόδους, που ήταν οι ακόλουθες:

Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της.

Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της.

Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας.

Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας.

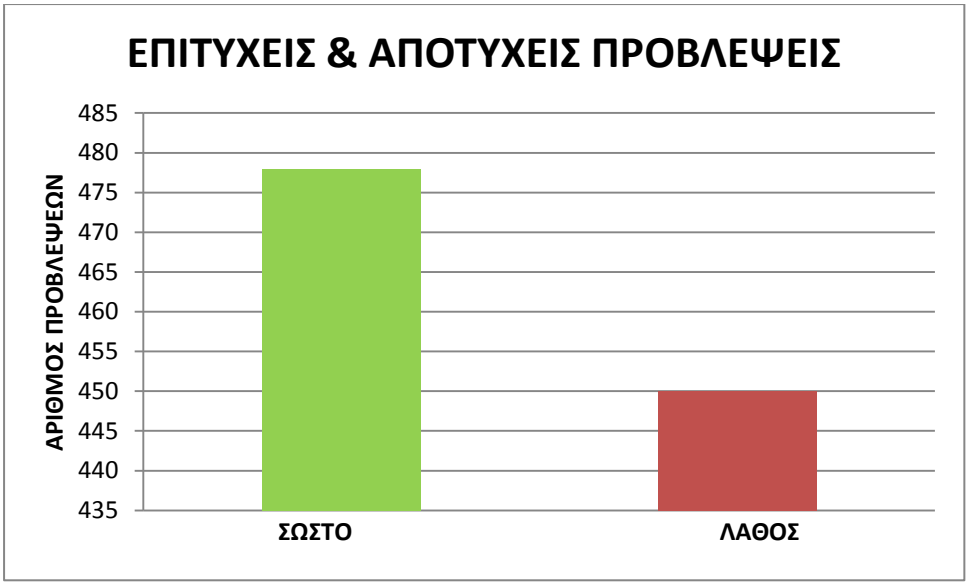
Μέσος όρος πόντων που πέτυχε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες.
Μέσος όρος πόντων που δέχτηκε η γηπεδούχος ομάδα στην έδρα της, τους τελευταίους έξι αγώνες.
Μέσος όρος πόντων που πέτυχε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες.
Μέσος όρος πόντων που δέχτηκε η φιλοξενούμενη ομάδα εκτός έδρας, τους τελευταίους έξι αγώνες.

Χρησιμοποιήσαμε 4 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 100. Το ποσοστό επιτυχίας που πήραμε ανήλθε στο 50%.

Πέμπτη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 8
Νευρώνες Κρυφού Επιπέδου: 8
Νευρώνες Εξόδου: 1
Ρυθμός Μάθησης: 0.5
Σίγμα: 0.2
Περιόδοι Εκπαίδευσης: 2004-2013
Κατώφλι:200 πόντοι
Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.52

Στην πέμπτη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 8 εισόδους, τις 8 που αναφέραμε πιο πάνω. Διατηρήσαμε όλες τις προηγούμενες παραμέτρους, με διαφορά ότι αυξήσαμε τους νευρώνες του κρυφού επιπέδου σε 8. Το ποσοστό επιτυχίας που πήραμε ήταν λίγο καλύτερο και ανήλθε στο 52%.

Έκτη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 8

Νευρώνες Κρυφού Επιπέδου: 16

Νευρώνες Εξόδου: 1

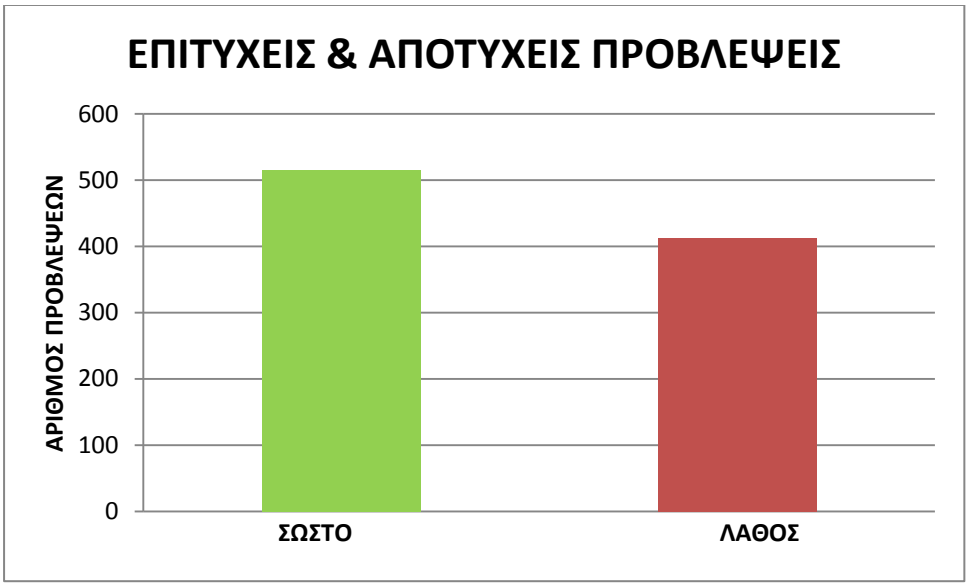
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.55

Στην έκτη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 8 εισόδους, τις 8 που αναφέραμε πιο πάνω. Διατηρήσαμε όλες τις προηγούμενες παραμέτρους, με διαφορά ότι αυξήσαμε τους νευρώνες του κρυφού επιπέδου σε 16. Το ποσοστό επιτυχίας αυξήθηκε και μπορούμε να συμπεράνουμε ότι όταν οι είσοδοι ήταν 8, η αύξηση των αριθμών των νευρώνων του κρυφού επιπέδου, βοήθησε στην καλύτερη πρόβλεψη των αποτελεσμάτων. Το ποσοστό επιτυχίας που πετύχαμε ανήλθε στο 55%.

Έβδομη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 10

Νευρώνες Κρυφού Επιπέδου: 10

Νευρώνες Εξόδου: 1

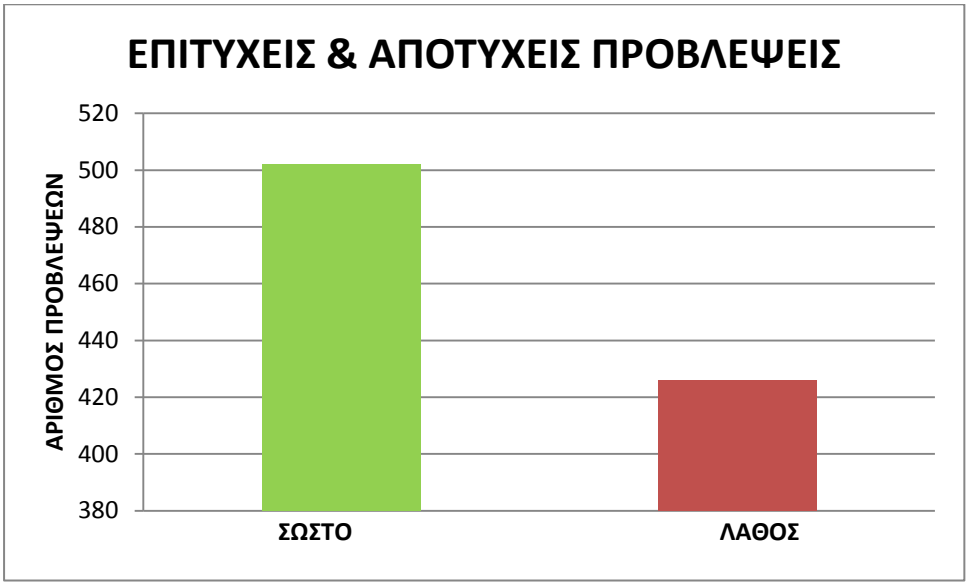
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.54

Στην έβδομη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 10 εισόδους, τις πρώτες 10 που αναφέραμε στα δεδομένα εισόδου. Χρησιμοποιήσαμε 10 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 1000. Τα αποτελέσματα που πήραμε ήταν καλύτερα από κάποιες από τις προηγούμενες εκτελέσεις και έφτασαν το 54%.

Όγδοη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 10

Νευρώνες Κρυφού Επιπέδου: 20

Νευρώνες Εξόδου: 1

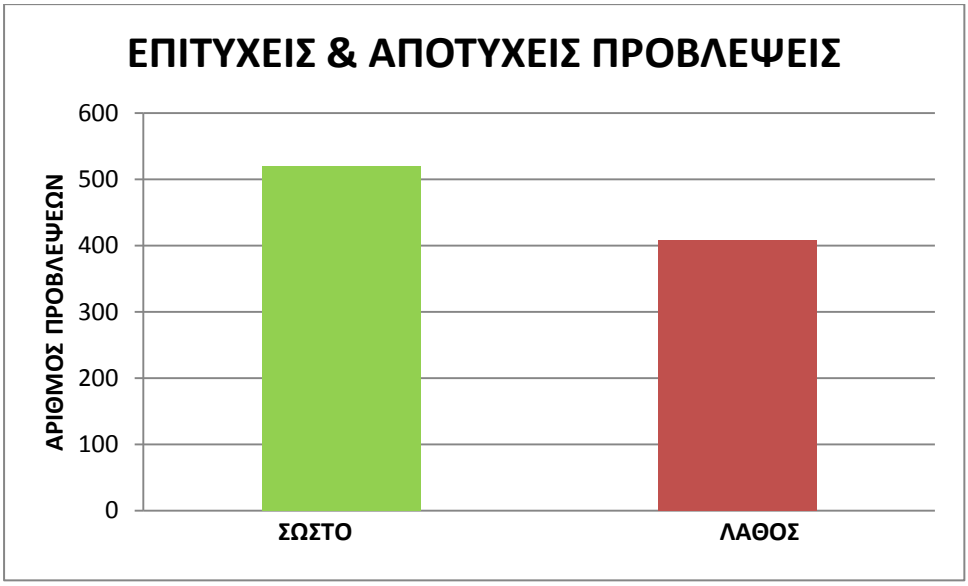
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Περιόδοι Εκπαίδευσης: 2004-2013

Κατώφλι:200 πόντοι

Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.56

Στην όγδοη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 10 εισόδους, τις πρώτες 10 που αναφέραμε στα δεδομένα εισόδου. Χρησιμοποιήσαμε 20 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν όλοι οι περίοδοι εκπαίδευσης που είχαμε στη διάθεσή μας, ενώ ο αριθμός των επαναλήψεων ήταν 100. Τα αποτελέσματα που πήραμε ήταν καλύτερα από κάποιες από τις προηγούμενες εκτελέσεις και έφτασαν το 56% που ήταν και το πιο ψηλό ποσοστό όσο αφορά την υλοποίηση με δίκτυα RBF.

Ένατη εκτέλεση:

Δεδομένα:

Νευρώνες Εισόδου: 10

Νευρώνες Κρυφού Επιπέδου: 20

Νευρώνες Εξόδου: 1

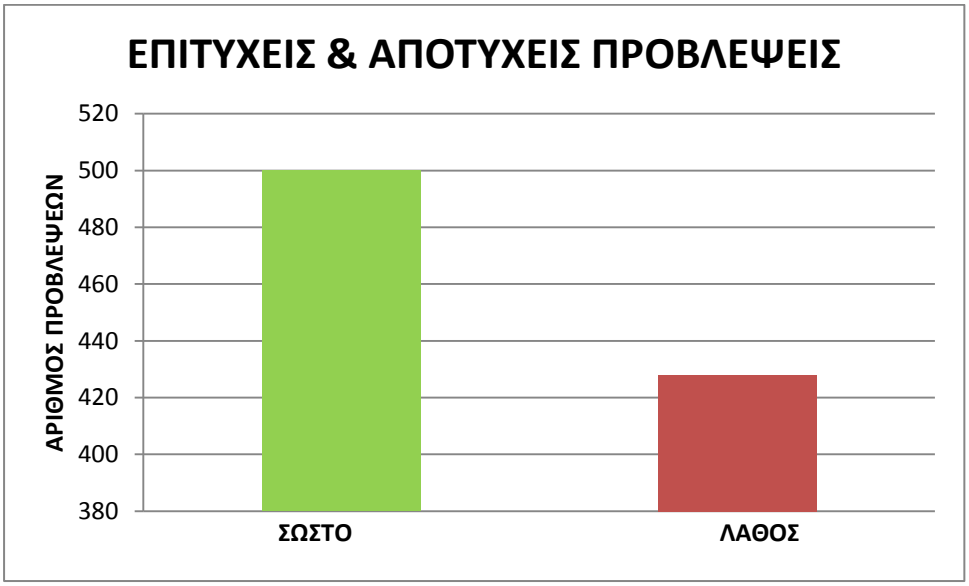
Ρυθμός Μάθησης: 0.5

Σίγμα: 0.2

Κατώφλι:200 πόντοι

Περίοδοι εκπαίδευσης: 2008-2013

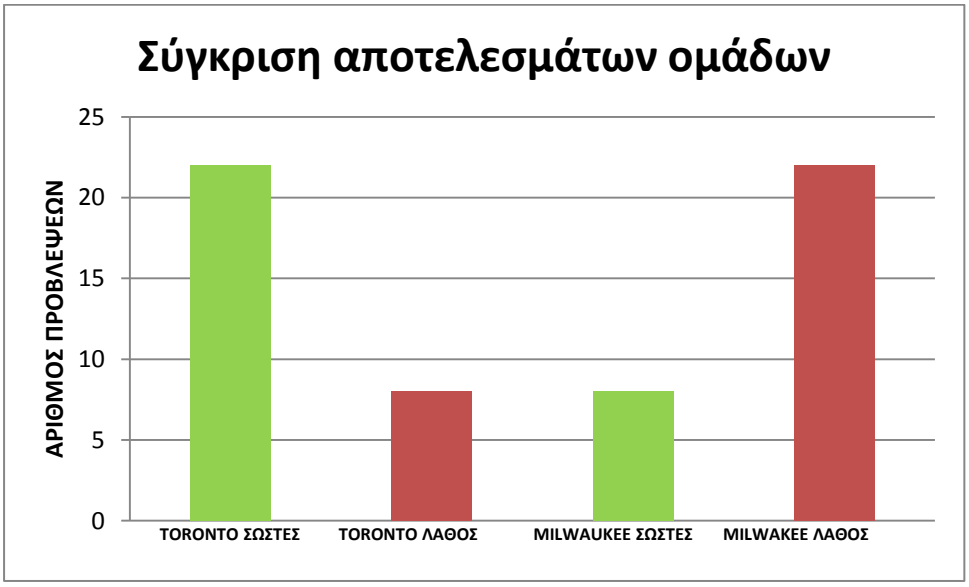
Επαναλήψεις:100



Ποσοστό επιτυχίας: 0.54

Στην ένατη εκτέλεση του αλγόριθμου RBF, χρησιμοποιήσαμε 10 εισόδους, τις πρώτες 10 που αναφέραμε στα δεδομένα εισόδου. Χρησιμοποιήσαμε 20 νευρώνες στο κρυφό επίπεδο. Ο ρυθμός μάθησης ήταν 0.5 ενώ το σίγμα 0.2. Χρησιμοποιήθηκαν οι περίοδοι εκπαίδευσης 2008-2013 για να δούμε αν με λιγότερες σεζόν θα παίρναμε καλύτερα αποτελέσματα. Κάτι τέτοιο δεν φάνηκε και το ποσοστό ήταν λίγο χαμηλότερο από την προηγούμενη εκτέλεση με 10 εισόδους και 20 κρυφούς νευρώνες. Το ποσοστό έφτασε το 54%.

Σύγκριση αποτελεσμάτων ομάδων



Όπως και στην περίπτωση με τον αλγόριθμο back propagation συγκρίναμε τις δύο ομάδες που είχαν τα καλύτερα και τα χειρότερα αποτελέσματα. Παρατηρούμε ότι η ομάδα Toronto είχε σωστές τις 22 από τις 30 προβλέψεις ενώ η Milwaukee ήταν και πάλι η ομάδα με τις λιγότερες επιτυχημένες προβλέψεις. Αυτό όπως είπαμε οφείλεται στα πολλά σκαμπανεβάσματα των σκορ στους αγώνες της. Αν την αποκλείσουμε από τις επιλογές μας για το στοίχημα, μπορούμε να έχουμε μεγαλύτερο κέρδος.

Κεφάλαιο 8

Συζήτηση, Συμπεράσματα και μελλοντική εργασία

8.1 Συμπεράσματα

Σε αυτό το σημείο θα περιγράψουμε περιληπτικά τα συμπεράσματα που προέκυψαν από τα αποτελέσματα των δύο αλγόριθμων, τι σημαίνουν αυτά τα αποτελέσματα, κατά πόσο είναι κερδοφόρα και πως μπορούμε να επεκτείνουμε και να βελτιώσουμε τα δίκτυα.

Αρχικά μπορούμε να πούμε ότι η επιλογή να προβλέψουμε τον συνολικό αριθμό πόντων μπορεί να χαρακτηριστεί επιτυχημένη. Οι αποδόσεις που παίρνουμε με αυτή την επιλογή είναι της τάξης 1.90 προς ένα για δύο αποτελέσματα που έχουν τις ίδιες πιθανότητες επαλήθευσης. Δεν μπορούσαμε να έχουμε καλύτερη απόδοση από αυτή, λόγω της γκανιότας που παρακρατάνε τα πρακτορεία στοιχημάτων. Όμως, αφού έχουμε πετύχει ορθές προβλέψεις για 70% από αυτά τα αποτελέσματα, το κέρδος είναι δεδομένο.

Αν υποθέσουμε ότι ποντάραμε 100 ευρώ σε 100 στοιχήματα του ενός ευρώ το καθένα, τότε έχουμε κερδίσει 133 ευρώ. Δηλαδή μιλάμε για καθαρό κέρδος 33,% κάτι που είναι πολύ μεγάλο αν έχουμε σκοπό να επενδύσουμε σοβαρά χρήματα.

Πρέπει να αναφέρουμε ότι εξαιτίας των πολλών κατωφλίων, οι δοκιμές που κάναμε κάθε φορά, αφορούσαν διαφορετικό κατώφλι. Στην περίπτωση όπου πετύχαμε 70% επιτυχία με τον αλγόριθμο Back-Propagation, το κατώφλι που χρησιμοποιήθηκε ήταν 200 πόντοι. Ασφαλώς δεν ήταν αυτό που θα χρησιμοπούταν σε όλους τους αγώνες

που αποτελούσαν τα δεδομένα ελέγχου. Δεδομένου όμως ότι η επιτυχία μας ήταν περίπου η ίδια και με διαφορετικά κατώφλια, τα οποία κάλυπταν όλα τα πιθανά, θεωρούμε ότι το κέρδος μας είναι πολύ κοντά στο ποσοστό που αναφέρουμε. Πιθανόν να είναι λίγο μικρότερο από 33% αφού κάποιες προβλέψεις μπορεί να είναι σωστές για 200 πόντους, όμως όχι για το ακριβές κατώφλι. Όμως μπορούμε να χρησιμοποιήσουμε εναλλακτικές επιλογές πονταρισμάτων που προσφέρουν οι εταιρίες και να πάρουμε μικρότερη απόδοση ή ακόμη και μεγαλύτερη.

Ο μέσος όρος της απόκλισης μας για τον αλγόριθμο Back-Propagation, ανήλθε στους 14 πόντους ανά παιχνίδι. Αυτό, θεωρούμε ότι είναι απόλυτα επιτυχές, δεδομένου ότι κάποια παιχνίδια όπου έχουμε μεγάλη απόκλιση και συντίνουν στο να αυξάνεται αυτός ο αριθμός, είναι ούτως ή άλλως χαμένα για μας. Αυτό σημαίνει ότι είμαστε πολύ κοντά σε αρκετά από τα παιχνίδια που προβλέπουμε.

Αντίθετα, στον αλγόριθμο με τα RBF δίκτυα, ο μέσος όρος απόκλισης έφτασε τους 25 πόντους κάτι που δεν το θεωρούμε ικανοποιητικό. Αυτό σημαίνει ότι η απόκλιση μας είναι μεγάλη σε αρκετά παιχνίδια και τα RBF θα πρέπει να βελτιωθούν.

Η επιλογή μας, να χρησιμοποιήσουμε το NBA σαν το πρωτάθλημα του οποίου θα προβλέπουμε τα αποτελέσματα, μπορεί επίσης να χαρακτηριστεί επιτυχημένη. Αυτό, λόγω του ότι είναι το μεγαλύτερο πρωτάθλημα στον κόσμο, τόσο σε αριθμό αγωνιστικών, όσο και σε χρήματα που δαπανώνται για τον καταρτισμό των ρόστερ των ομάδων. Οι χορηγίες που προσφέρονται στις ομάδες είναι τεράστιες, κάτι που καθιστά το NBA ένα αξιόπιστο πρωτάθλημα, χωρίς προκατάληψη εναντίων διαιτητών ή ομάδων που μπορεί να αγοράζουν ή να πουλούν παιχνίδια. Όπως και στην περίπτωση του ποδοσφαίρου, τα άτομα που ασχολήθηκαν με το θέμα αυτό, προτίμησαν πρωταθλήματα όπου δεν παρουσιάζονταν παράξενα αποτελέσματα, πρωταθλήματα που δεν τιθόταν αμφισβήτηση για την εγκυρότητά τους[9]. Ακόμη, αποτελεί το πρωτάθλημα με τον μεγαλύτερο τζίρο σε στοιχήματα αγώνων καλαθόσφαιρας και το ιστορικό των αποτελεσμάτων είναι εύκολα προσπελάσιμο για κάποιον που θέλει να αναζητήσει την προϊστορία των ομάδων.

Τα δεδομένα εισόδου που χρησιμοποιήσαμε, βασίζονταν σε παλιά αποτελέσματα. Δεν υπολογίσαμε τραυματισμούς παικτών, τιμωρίες έδρας και κίνητρο. Όμως, όπως αποδείχτηκε αυτό δεν κατέστη εμπόδιο στην επιτυχή λειτουργία του δικτύου μας, αφού τα αποτελέσματα ήταν επιτυχημένα. Ασφαλώς αν υπολογίζαμε και αυτές τις

παραμέτρους, ίσως να είχαμε καλύτερα αποτελέσματα. Ίσως και όχι όμως, αφού έτσι θα αυξάναμε την πολυπλοκότητα του δικτύου κάτι που θα ήταν εις βάρος μας. Το σίγουρο είναι όμως ότι, θα είχαμε περισσότερες επιλογές για να τροφοδοτήσουμε το δίκτυο, άρα και περισσότερες πιθανότητες επιτυχίας.

Μπορούμε να πούμε με βεβαιότητα ότι δεν καταλήξαμε σε κάποιο κανόνα που να μας λέει ότι κάποια παράμετρος είναι απαραίτητη για την ομαλή λειτουργία του δικτύου. Τα αποτελέσματα προέκυψαν, βασίζονται στις συνολικές παραμέτρους και δεν μπορούμε να προσδιορίσουμε αν κάποια συγκεκριμένη κάνει αισθητή διαφορά. Αυτό που παρατηρήσαμε όμως, όσον αφορά τον αλγόριθμο Back-Propagation, είναι ότι όταν αυξήσαμε τους κρυφούς νευρώνες στο πρώτο κρυφό επίπεδο και ο αριθμός τους έγινε περίπου ο διπλάσιος από αυτούς του επιπέδου εισόδου, τα αποτελέσματα ήταν καλύτερα.

Ο αριθμός των επαναλήψεων που χρησιμοποιούνται στην διαδικασία εκπαίδευσης παίζει σημαντικό ρόλο αφού παρατηρούμε ότι όσες περισσότερες δίνουμε, μέχρις ενός σημείου, το δίκτυο πετυχαίνει καλύτερα αποτελέσματα. Ασφαλώς, θα πρέπει να υπάρχει ένα όριο ως προς αυτό τον αριθμό, γιατί ένας πολύ μεγάλος αριθμός επαναλήψεων μπορεί να φέρει αντίθετα αποτελέσματα από αυτά που αναμένουμε.

Μεγάλη έμφαση δόθηκε στον αριθμό των σεζόν που θα χρησιμοποιούσαμε. Υπήρχε ο ενδοιασμός για το αν θα χρησιμοποιούσαμε αποτελέσματα πολύ παλαιών σεζόν, ή αν θα ήταν καλύτερο να χρησιμοποιήσουμε μόνο από πολύ πρόσφατες. Έγιναν πολλές δοκιμές για το πόσες σεζόν θα χρησιμοποιούσαμε και καταλήξαμε στο συμπέρασμα ότι οι δέκα τελευταίες, ήταν ο ιδανικός αριθμός για να πετύχουμε ένα κερδοφόρο δίκτυο. Οι λιγότερες σεζόν μας απέφεραν λίγο πιο μικρό κέρδος. Αν δοκιμάζαμε να χρησιμοποιήσουμε περισσότερες από δέκα σεζόν ίσως να παίρναμε καλύτερα, αλλά ίσως και χειρότερα αποτελέσματα. Εξαρτάται από το αν στις προηγούμενες σεζόν τα αποτελέσματα ήταν παρόμοια με αυτά που χρησιμοποιήσαμε ή αν ήταν πολύ διαφορετικά.

Δεν πήραμε τα αναμενόμενα αποτελέσματα όσον αφορά τον αλγόριθμο με τα RBF δίκτυα. Αυτό ίσως να οφείλεται στη φύση του προβλήματος, ίσως και όχι. Δυστυχώς δεν υπήρξε ο απαραίτητος χρόνος για να προσπαθήσουμε να τον βελτιώσουμε. Παρόλα αυτά, είμαστε ικανοποιημένοι από το αποτέλεσμα που πετύχαμε

χρησιμοποιώντας τον αλγόριθμο Back-Propagation, αφού το 70% επιτυχίας, μας αποφέρει υψηλό ποσοστό κέρδους.

Καταληκτικά, αισθάνομαι απόλυτα ικανοποιημένος από το αποτέλεσμα που πετύχαμε, τουλάχιστον στον αλγόριθμο Back-Propagation. Η μεγάλη επιτυχία του Back-Propagation, μας αφήνει ένα μεγάλο ποσοστό κέρδους, κάτι που ήταν και το ζητούμενο. Μπορώ να πω ότι δεν υπάρχουν και μεγάλα περιθώρια βελτίωσης του αποτελέσματος του αλγόριθμου αυτού. Όσον αφορά τα RBF δίκτυα, θα πρέπει να τα βελτιώσουμε, αφού δεν δικαιολογήται τόσο μεγάλη διαφορά των δύο αλγόριθμων.

8.2 Μελλοντική Εργασία

Με το πέρας της διπλωματικής αυτής εργασίας και με βάση τα αποτελέσματα τα οποία προέκυψαν από αυτή, προκύπτει ότι πιθανών να υπάρχουν ακόμη περιθώρια βελτίωσης των αποτελεσμάτων και αύξησης των ποσοστών επιτυχίας στο πρόβλημα πρόβλεψης αποτελεσμάτων αγώνων NBA με τη χρήση νευρωνικών δικτύων.

Καταρχάς μπορούμε να πούμε ότι αν χρησιμοποιήσουμε περισσότερες παραμέτρους ναι μεν θα αυξηθεί περισσότερο η πολυπλοκότητα, όμως μπορούμε να δουλέψουμε πάνω σε αυτές και να χρησιμοποιήσουμε τις κατάλληλες, μέσα από ένα μεγαλύτερο εύρος επιλογών.

Ακόμη μπορούμε να εστιάσουμε στο ποιες ομάδες, έχουν μεγαλύτερα ποσοστά σωστών προβλέψεων και να ασχοληθούμε μόνο με αυτές όσο αφορά το στοίχημα, ώστε να περιορίσουμε τις απώλειες μας από τα χαμένα πονταρίσματα. Με αυτό τον τρόπο το κέρδος θα μπορεί να αυξηθεί.

Από τα αποτελέσματα που πήραμε, έχουμε μια σχετική εικόνα ότι τα νευρωνικά δίκτυα μπορούν να μας αποφέρουν κέρδος αν τα χρησιμοποιήσουμε για το πρόβλημα αυτό. Όμως δεν είμαστε σε θέση να πούμε ότι αποδείξαμε ότι δουλεύουν σε βάθος χρόνου ή ότι δεν δουλεύουν. Σαν μελλοντική εργασία έχω σκοπό να ερευνήσω κατά πόσο αποτελούν αξιόπιστη πηγή πληροφόρησης για προβλέψεις αγώνων.

Πέρα από τους δύο αλγόριθμους που χρησιμοποιήσαμε, υπάρχουν και άλλα εργαλεία που βασίζονται στα νευρωνικά δίκτυα και ίσως μπορούν να αποφέρουν καλύτερα αποτελέσματα.

Ένα από αυτά τα εργαλεία είναι τα Support Vector Machines. Μπορούμε να χρησιμοποιήσουμε ένα SVM simulator και να δούμε αν τα αποτελέσματα είναι καλύτερα ή ακόμα και να κατασκευάσουμε ένα SVM για το συγκεκριμένο πρόβλημα.

Πέρα από αυτά, έχω σκοπό να επεκτείνω τα δίκτυα και να γίνει δοκιμή για πρόβλεψη άλλων αποτελεσμάτων, όπως η νίκη μιας ομάδας με διαφορά ενός αριθμού πόντων, ή η πρόβλεψη αποτελεσμάτων στα 12λεπτα των αγώνων. Ίσως αυτού του είδους τα στοιχήματα να είναι πιο κερδοφόρα από το OVER/UNDER που ήταν το κομμάτι που υλοποιήσαμε. Σημασία για εμάς έχει η επίτευξη κέρδους, γι αυτό και δεν μας απασχολεί το είδος του στοιχήματος. Ασχοληθήκαμε με το OVER/UNDER, λόγω του ότι σε αυτό το είδος υπήρχε προηγούμενη δουλειά για ποδοσφαιρικά στοιχήματα και ήταν πιο εύκολο να βασιστούμε στα προηγούμενα λάθη ώστε να μην τα επαναλάβουμε.

Τέλος μια καλή ιδέα θα ήταν να δοκιμάσουμε να αυξήσουμε τους νευρώνες του επιπέδου εξόδου και να μεταβάλουμε τον τρόπο που κωδικοποιείται η έξοδος, ώστε κάθε φορά που παίρνουμε μια πρόβλεψη να μας δίδεται και η πιθανότητα επαλήθευσής της. Με αυτό τον τρόπο θα βλέπουμε ποιες προβλέψεις έχουν μεγαλύτερη πιθανότητα να επαληθευτούν και θα επιλέγουμε μόνο αυτές για το ποντάρισμα.

Δεν μπορούμε να προσδιορίσουμε τι από όλα αυτά θα φέρει καλύτερα αποτελέσματα και τι χειρότερα. Πρέπει να γίνουν οι απαραίτητες υλοποιήσεις και δοκιμές και αυτό είναι κάτι που θα με ενδιέφερε ιδιαίτερα να ασχοληθώ.

Βιβλιογραφία – Αναφορές

[1] Θεοφάνης Νεοκλέους , Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου 2007

[2] Ανδρέας Ιακώβου, Ατομική Διπλωματική Εργασία, Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου 2010

[3] Υλικό Διαλέξεων Υπολογιστικών Συστημάτων Μάθησης (ΕΠΛ 442), Πανεπιστήμιο Κύπρου, Δρ. Χρίστος Χριστοδούλου

[4] Donald O. Hebb, 1949, The Organization of Behavior

[5] Widrow, Hoff 1962, Learning Rule

[6] Paul Werbos 1974, Thesis

[7] Schölkopf Bernhard, Alex Smola, "Learning with Kernels", MIT Press, Cambridge, MA, 2002

[8]Donald R.Tveter, *The Backprop Algorithm*, www.donveter.com

[9] Goddard, J. and Asimakopoulos, I. (2004). Modelling football match results and the efficiency of fixed-odds betting. *Journal of Forecasting*. http://stat-www.berkeley.edu/~aldous/157/Papers/goddard.pdf

[10] www.basketball-reference.com

[11] www.wikipedia.org

[12] www.bet365.com

ΠΑΡΑΡΤΗΜΑ Α – Υλοποίηση Back-Propagation

Κλάση Tdata.Java

```
public class Tdata {

	double input1;
	double input2;
	double input3;
	double input4;
	double input5;
	double input6;
	double output;
	public Tdata(double input1, double input2, double input3, double input4,
			double input5, double input6, double output) {
		super();
		this.input1 = input1;
		this.input2 = input2;
		this.input3 = input3;
		this.input4 = input4;
		this.input5 = input5;
		this.input6 = input6;
		this.output = output;
	}


}
```

Κλάση ReadFromFile.Java

```
import java.io.BufferedReader;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.Reader;

public class ReadFromFile {



	public void readFromFile(String filename) {
		BufferedReader in = null;

		try {
			in = new BufferedReader(new FileReader(filename));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		String line = "";
		String[] temp;
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		temp = line.split(" ");
```

```
                CalculateScore.numHL1N = Integer.parseInt(temp[1]);

                if (CalculateScore.numHL1N < 0) {
                        System.err
                                        .println("Wrong parameter count for
numHiddenLayerOne Neurons. Exit ...");
                        System.exit(-1);
                }

                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }

                temp = line.split(" ");
                CalculateScore.numHL2N = Integer.parseInt(temp[1]);
                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }
                if (CalculateScore.numHL2N < 0) {
                        System.err
                                        .println("Wrong parameter count for
numHiddenLayerTwo Neurons. Exit ...");
                        System.exit(-1);
                }
                if (CalculateScore.numHL1N == 0 && CalculateScore.numHL2N > 0) {
                        System.err
                                        .println("Wrong parameter count for
numHiddenLayerTwoNeurons\nYou can not have hidden layer 2 and no hidden layer
1.Exit..");
                        System.exit(-1);
                }
```

```
                temp = line.split(" ");
                CalculateScore.numInputNeurons = Integer.parseInt(temp[1]);
                if (CalculateScore.numInputNeurons <= 0) {
                        System.err
                                        .println("Wrong number of parameters for
numHiddenLayerTwoNeurons.Exit..");
                        System.exit(-1);
                }
                try {
                        line = in.readLine();
                } catch (IOException e) {
                                                e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.numOutNeurons = Integer.parseInt(temp[1]);
                if (CalculateScore.numOutNeurons <= 0) {
                        System.err
                                        .println("Wrong number of parameters for
numHiddenLayerTwoNeurons.Exit..");
                        System.exit(-1);
                }
                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.learningRate = Double.parseDouble(temp[1]);
                if (CalculateScore.learningRate < 0.0 || CalculateScore.learningRate > 1.0) {
                        System.err
                                        .println("Wrong number of parameters for
learningRate.\nMust be between 0 and 1.Exit..");
                        System.exit(-1);
                }
```

```
                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.momentum = Double.parseDouble(temp[1]);
                if (CalculateScore.momentum <= 0.0 || CalculateScore.momentum >= 1.0) {
                        System.err
                                        .println("Wrong number of parameters for
momentum.\nMust be between 0 and 1 (inclusive).Exit..");
                        System.exit(-1);
                }
                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.maxIterations = Integer.parseInt(temp[1]);
                try {
                        line = in.readLine();
                } catch (IOException e) {

                        e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.trainFile = new String(temp[1]);
                try {
                        line = in.readLine();
                } catch (IOException e) {
                        e.printStackTrace();
                }
                temp = line.split(" ");
                CalculateScore.testFile = new String(temp[1]);
```

```
	}

	public void readFromFile(String filename, boolean f) {
		Reader reader = null;
		try {
			reader = new InputStreamReader(new FileInputStream(filename));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		}
		BufferedReader in = new BufferedReader(reader);
		String line = "";
		String[] temp;
		try {
			line = in.readLine();
		} catch (IOException e) {
			e.printStackTrace();
		}
		while (line != null) {

			temp = line.split(" ");
			Tdata tmp = new Tdata(Double.parseDouble(temp[0]),
Double.parseDouble(temp[1]), Double.parseDouble(temp[2]),
					Double.parseDouble(temp[3]),
Double.parseDouble(temp[4]), Double.parseDouble(temp[5]),
Double.parseDouble(temp[6]));
			if (f)
				CalculateScore.trainData.add(tmp);
			try {
				line = in.readLine();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
	}
```

```
        public static void main(String[] args) {
                ReadFromFile x = new ReadFromFile();
                x.readFromFile("parameters.txt");
                x.readFromFile("training.txt", true);
        }

}
```

Κλάση PrintInFile.Java

```
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.PrintStream;
import java.util.ArrayList;

public class PrintInFile {

        public void printWeights(String filename, ArrayList<InputLayer> inlayer,
                        ArrayList<HiddenLayer> hiddenLayer1,
                        ArrayList<HiddenLayer> hiddenLayer2) {

                FileOutputStream fout;
                try {
                        fout = new FileOutputStream(filename);
                        PrintStream p = new PrintStream(fout);
                        p.println("                Weights after training");
                        p.println();
                        p.println();
                        p.println("Weights for input layer:");
                        p.println("------------------------");
                        String x;
                        String bias;
                        if (CalculateScore.numHL1N == 0)
                                x = new String("output");
                        else
```

```
                    x = new String("hidden");
                for (int i = 0; i < inlayer.size(); i++) {
                    if (i == (inlayer.size() - 1))
                        bias = new String(" (bias) ");
                    else
                        bias = new String(" ");
                    p.println("Neuron " + inlayer.get(i).num + bias);
                    p.println("-----------------");
                    for (int j = 0; j < inlayer.get(i).weights.size(); j++)
                        p.println("Neuron " + inlayer.get(i).num + bias
                                + "has weight with neuron " + j + "
of " + x
                                + " layer = " +
inlayer.get(i).weights.get(j));
                    p.println();

                }
                p.println();
                p
                        .println("----------------------------------------------------
--------------");
                p.println();
                if (CalculateScore.numHL1N != 0) {
                    p.println("Weights for hidden layers:");
                    p.println("------------------------");

                    p.println("Hidden Layer 1 ");
                    p.println("-------------------");
                    if (CalculateScore.numHL2N == 0)
                        x = new String("output");
                    else
                        x = new String("hidden");
                    for (int i = 0; i < hiddenLayer1.size(); i++) {
                        if (i == (hiddenLayer1.size() - 1))
                            bias = new String(" (bias) ");
                        else
                            bias = new String(" ");
```

```
                                        p.println("Neuron " + hiddenLayer1.get(i).num +
bias);
                                        p.println("-----------------");
                                        for (int j = 0; j < hiddenLayer1.get(i).weights.size();
j++) {
                                                p.println("Neuron " +
hiddenLayer1.get(i).num + bias
                                                        + "has weight with neuron "
+ j + " of " + x
                                                        + " layer = "
                                                        +
hiddenLayer1.get(i).weights.get(j));
                                        }
                                        p.println();
                                }
                                p.println();
                                p.println();
                        }
                        if (CalculateScore.numHL2N != 0) {

                                p.println("Hidden Layer 2 ");
                                p.println("-------------------");

                                x = new String("output");

                                for (int i = 0; i < hiddenLayer2.size(); i++) {
                                        if (i == (hiddenLayer2.size() - 1))
                                                bias = new String(" (bias) ");
                                        else
                                                bias = new String(" ");
                                        p.println("Neuron " + hiddenLayer2.get(i).num +
bias);
                                        p.println("-----------------");
                                        for (int j = 0; j < hiddenLayer2.get(i).weights.size();
j++) {
                                                p.println("Neuron " +
hiddenLayer2.get(i).num + bias
```

```
                                                    + "has weight with neuron "
+ j + " of " + x
                                                    + " layer = "
                                                    +
hiddenLayer2.get(i).weights.get(j));
                                        }
                                        p.println();
                                }
                                p.println();
                                p.println();
                        }
                        fout.close();
                }
                catch (IOException e) {
                        System.err.println("Unable to write to file");
                        System.exit(-1);
                }
        }

        public void printResults(String filename, ArrayList<Double> trainError) {

                FileOutputStream fout;
                try {
                        fout = new FileOutputStream(filename);
                        PrintStream p = new PrintStream(fout);
                        p.println("Iteration\Training error\t");
                        p.println("----------\t--------------\t");
                        for (int i = 0; i < trainError.size(); i++)
                                p.println("    " + i + "\t\t" + trainError.get(i));
                        fout.close();
                }
                catch (IOException e) {
                        System.err.println("Unable to write to file");
                        System.exit(-1);
                }
        }
```

```
}
```

6.1.5 Κλάση InputLayer

```
import java.util.ArrayList;

public class InputLayer {
	InputLayer(int num,ArrayList<Double> w){
		this.num=num;
		this.input=0;
		this.weights = new ArrayList<Double>(w);
		this.previous_weights = new ArrayList<Double>(w);
	}
	int num;
	double input;
	ArrayList<Double> weights ;
	ArrayList<Double> previous_weights;


	public void setWeight(int position, double w) {
		this.weights.set(position, w);
	}


	public void setPreviousWeight(int position, double w) {
		this.previous_weights.set(position, w);
	}

	public void setInput(double input){
		this.input=input;
	}
}
```

6.1.6 Κλάση HiddenLayer.Java

```
import java.util.ArrayList;

public class HiddenLayer {
	HiddenLayer(int num, int layerNum, ArrayList<Double> w) {
		this.num = num;
		this.layerNum=layerNum;
		this.actValue = 0;
		this.weights = new ArrayList<Double>(w);
		this.previous_weights = new ArrayList<Double>(w);
		this.error = 0;
	}

	int num;
	int layerNum;
	double actValue;
	double sum = 0;
	ArrayList<Double> weights = new ArrayList<Double>();
	ArrayList<Double> previous_weights = new ArrayList<Double>();
	double error;


	public void setWeight(int position, double w) {
		this.weights.set(position, w);
	}


	public void setPreviousWeight(int position, double w) {
		this.previous_weights.set(position, w);
	}
}
```

Κλάση Game

```
public class Game {
	public Game() {
		super();
	}
	public Game(String homeTeam, String awayTeam, int homeTeamScore,
			int awayTeamScore) {
		super();
		this.homeTeam = homeTeam;
		this.awayTeam = awayTeam;
		this.homeTeamScore = homeTeamScore;
		this.awayTeamScore = awayTeamScore;
	}
	public String getHomeTeam() {
		return homeTeam;
	}
	public void setHomeTeam(String homeTeam) {
		this.homeTeam = homeTeam;
	}
	public String getAwayTeam() {
		return awayTeam;
	}
	public void setAwayTeam(String awayTeam) {
		this.awayTeam = awayTeam;
	}
	public int getHomeTeamScore() {
		return homeTeamScore;
	}
	public void setHomeTeamScore(int homeTeamScore) {
		this.homeTeamScore = homeTeamScore;
	}
	public int getAwayTeamScore() {
		return awayTeamScore;
	}
	public void setAwayTeamScore(int awayTeamScore) {
```

```
            this.awayTeamScore = awayTeamScore;
        }
        public String homeTeam;
        public String awayTeam;
        public int homeTeamScore;
        public int awayTeamScore;


        public static Game createGameFromString(String s){

            Game g = new Game();
            int space = s.indexOf(" ");

            String homeTeam = s.substring(0, space);
            g.setHomeTeam(homeTeam);

            s = s.substring(space + 1);
            space = s.indexOf(" ");
            String awayTeam = s.substring(0, space);
            g.setAwayTeam(awayTeam);

            s = s.substring(space + 1);
            space = s.indexOf(" ");
            int homeTeamScore = Integer.valueOf(s.substring(0, space));
            g.setHomeTeamScore(homeTeamScore);

            s = s.substring(space + 1);
            space = s.indexOf(" ");
            int awayTeamScore = Integer.valueOf(s.substring(0,space));
            g.setAwayTeamScore(awayTeamScore);

            return g;
        }
}
```

Κλάση OutputLayer.Java

```
public class OutputLayer {
	OutputLayer(int num) {
		this.num = num;
		this.output = 0;
		this.sum = 0;
		this.error = 0;
	}

	int num;
	double output;
	double sum;;
	double totalError;
	double error;
}
```

Κλάση FeedForward

```
import java.util.ArrayList;

public class FeedForward {

	public void calculateSum_l1(ArrayList<InputLayer> il,
			ArrayList<HiddenLayer> hl) {
		for (int i = 0; i < (hl.size() - 1); i++) {
			hl.get(i).sum = 0;
			for (int j = 0; j < il.size(); j++) {
				hl.get(i).sum = hl.get(i).sum
					+ (il.get(j).input * il.get(j).weights.get(i));
			}
		}
	}
```

```
	public void calculateSum_hidden(ArrayList<HiddenLayer>
hl1,ArrayList<HiddenLayer> hl2) {
		for (int i = 0; i < (hl2.size() - 1); i++) {
			hl2.get(i).sum = 0;
			for (int j = 0; j < hl1.size(); j++) {
				hl2.get(i).sum = hl2.get(i).sum
					+ (hl1.get(j).actValue *
hl1.get(j).weights.get(i));
			}
		}
	}

	public void calculateSum_out2(ArrayList<InputLayer> il,
			ArrayList<OutputLayer> ol) {
		for (int i = 0; i < ol.size(); i++) {
			ol.get(i).sum = 0;
			for (int j = 0; j < il.size(); j++) {
				ol.get(i).sum = ol.get(i).sum
					+ (il.get(j).input * il.get(j).weights.get(i));
			}
		}
	}

	public void calculateSum_out(ArrayList<HiddenLayer> hl,
			ArrayList<OutputLayer> ol) {
		for (int i = 0; i < ol.size(); i++) {
			ol.get(i).sum = 0;
			for (int j = 0; j < hl.size(); j++) {
				ol.get(i).sum = ol.get(i).sum
					+ (hl.get(j).actValue *
hl.get(j).weights.get(i));
			}
		}
	}
```

```
	public void sigmoidHidden(ArrayList<HiddenLayer> hl) {
		for (int i = 0; i < (hl.size() - 1); i++)
			hl.get(i).actValue = 1 / (1 + Math.exp(-hl.get(i).sum));
		hl.get((hl.size() - 1)).actValue = 1;
	}

	public void sigmoidOut(ArrayList<OutputLayer> ol) {
		for (int i = 0; i < ol.size(); i++)
			ol.get(i).output = 1 / (1 + Math.exp(-ol.get(i).sum));
	}

}
```

Κλάση BackPropagation.Java

```
import java.util.ArrayList;

public class BackPropagation {

	public void calculateErrorOl(OutputLayer ol, double tout) {
		ol.error = ol.output * (1 - ol.output) * (tout - ol.output);
	}

	public void calculateErrorHFinal(HiddenLayer hl, ArrayList<OutputLayer> ol) {
```

```
                double sum = 0;
                sum = 0;
                for (int j = 0; j < ol.size(); j++)
                        sum = sum + ((hl.weights.get(j) * ol.get(j).error));
                hl.error = hl.actValue * (1 - hl.actValue) * (sum);
        }



        public void calculateErrorHl(HiddenLayer hl1, ArrayList<HiddenLayer> hl2) {
                double sum = 0;

                for (int j = 0; j < hl2.size(); j++)
                        sum = sum + ((hl1.weights.get(j) * hl2.get(j).error));
                hl1.error = hl1.actValue * (1 - hl1.actValue) * (sum);
        }

        public void calculateNewWeightsIl(HiddenLayer neuron,
                        ArrayList<InputLayer> il) {
                double weight = 0;
                for (int i = 0; i < il.size(); i++) {
                        weight = il.get(i).weights.get(neuron.num)
                                        + (CalculateScore.learningRate * neuron.error *
il.get(i).input)
                                        + (CalculateScore.momentum * (il.get(i).weights
                                                        .get(neuron.num) -
il.get(i).previous_weights
                                                        .get(neuron.num)));
                        il.get(i).setPreviousWeight(neuron.num,
il.get(i).weights.get(neuron.num));
                        il.get(i).setWeight(neuron.num, weight);
                }
        }



        public void calculateNewWeightsIO(ArrayList<InputLayer> il,
                        OutputLayer neuron) {
```

```
                double weight = 0;
                for (int i = 0; i < il.size(); i++) {
                        weight = il.get(i).weights.get(neuron.num)
                                        + (CalculateScore.learningRate * neuron.error *
il.get(i).input)
                                        + (CalculateScore.momentum * (il.get(i).weights
                                                        .get(neuron.num) -
il.get(i).previous_weights
                                                        .get(neuron.num)));
                        il.get(i).setPreviousWeight(neuron.num,
il.get(i).weights.get(neuron.num));
                        il.get(i).setWeight(neuron.num, weight);
                }
        }


        public void calculateNewWeightsHl(HiddenLayer neuron,
                        ArrayList<HiddenLayer> hl1) {
                double weight = 0;
                for (int i = 0; i < hl1.size(); i++) {
                        weight = hl1.get(i).weights.get(neuron.num)
                                        + (CalculateScore.learningRate * neuron.error *
hl1.get(i).actValue)
                                        + CalculateScore.momentum
                                        * (hl1.get(i).weights.get(neuron.num) -
hl1.get(i).previous_weights
                                                        .get(neuron.num));
                        hl1.get(i).setPreviousWeight(neuron.num,
hl1.get(i).weights.get(neuron.num));
                        hl1.get(i).setWeight(neuron.num, weight);
                }
        }


        public void calculateNewWeightsHFinal(ArrayList<HiddenLayer> hl,
                        OutputLayer neuron) {
```

```
                double weight = 0;
                for (int i = 0; i < hl.size(); i++) {
                        weight = hl.get(i).weights.get(neuron.num)
                                        + (CalculateScore.learningRate * neuron.error *
hl.get(i).actValue)
                                        + (CalculateScore.momentum * (hl.get(i).weights
                                                        .get(neuron.num) -
hl.get(i).previous_weights
                                                        .get(neuron.num)));
                        hl.get(i).setPreviousWeight(neuron.num,
hl.get(i).weights.get(neuron.num));
                        hl.get(i).setWeight(neuron.num, weight);
                }
        }



}
```

Κλάση GenerateTest

```
import java.io.File;
import java.util.ArrayList;

public class GenerateTest {

        static int absolutMaxScore = 0;
        public static double treshold = 200;


        public ArrayList<Game> createListHomeTeamTotalGamesAsHost() {
                ArrayList<Game> homeTeamTotalGamesAsHost = new
ArrayList<Game>();
                Game game = new Game();
                InputOutput io = new InputOutput();
```

```
ArrayList<String> textInFile = new ArrayList<String>();


try{
        textInFile = io.readFileAsResource("resources/0405/"
                + MainClass.HOME_TEAM + ".txt");
}
catch(Exception e){};

try{
        textInFile.addAll(io.readFileAsResource("resources/0506/"
                + MainClass.HOME_TEAM + ".txt"));
}
catch(Exception e){};
try{
        textInFile.addAll(io.readFileAsResource("resources/0607/"
                        + MainClass.HOME_TEAM + ".txt"));
}
        catch(Exception e){};
        try{

        textInFile.addAll(io.readFileAsResource("resources/0708/"
                        + MainClass.HOME_TEAM + ".txt"));
        }
        catch(Exception e){};
        try{

        textInFile.addAll(io.readFileAsResource("resources/0809/"
                        + MainClass.HOME_TEAM + ".txt"));
        }
        catch(Exception e){};
        try{

        textInFile.addAll(io.readFileAsResource("resources/0910/"
                        + MainClass.HOME_TEAM + ".txt"));
        }
        catch(Exception e){};
```

```
                try{

                    textInFile.addAll(io.readFileAsResource("resources/1011/"
                            + MainClass.HOME_TEAM + ".txt"));
            }
            catch(Exception e){};
            try{

                    textInFile.addAll(io.readFileAsResource("resources/1213/"
                            + MainClass.HOME_TEAM + ".txt"));
            }catch(Exception e){};
            for (String s : textInFile) {
                    if (s.startsWith(MainClass.HOME_TEAM)) {
                            game = Game.createGameFromString(s);
                            homeTeamTotalGamesAsHost.add(game);
                    }
            }
            return homeTeamTotalGamesAsHost;
    }


    public ArrayList<Game> createListAwayTeamTotalGamesAway() {
            ArrayList<Game> awayTeamTotalGamesAsVisitor = new
ArrayList<Game>();
            Game game = new Game();
            InputOutput io = new InputOutput();
            ArrayList<String> textInFile = new ArrayList<String>();

            try {
                    textInFile = io.readFileAsResource("resources/0405/"
                                    + MainClass.AWAY_TEAM + ".txt");
            }
            catch(Exception e){};

            try{
```

```
            textInFile.addAll(io.readFileAsResource("resources/0506/"
                    + MainClass.AWAY_TEAM + ".txt"));
        }
        catch(Exception e){};

        try{

            textInFile.addAll(io.readFileAsResource("resources/0607/"
                    + MainClass.AWAY_TEAM + ".txt"));
        }
        catch(Exception e){};

        try{

            textInFile.addAll(io.readFileAsResource("resources/0708/"
                    + MainClass.AWAY_TEAM + ".txt"));
        }
        catch(Exception e){};

        try{

            textInFile.addAll(io.readFileAsResource("resources/0809/"
                    + MainClass.AWAY_TEAM + ".txt"));
        }
        catch(Exception e){};

        try{

            textInFile.addAll(io.readFileAsResource("resources/0910/"
                    + MainClass.AWAY_TEAM + ".txt"));
        }
        catch(Exception e){};

        try{

            textInFile.addAll(io.readFileAsResource("resources/1011/"
                    + MainClass.AWAY_TEAM + ".txt"));
```

```
        } catch (Exception e) {}
        ;

        try{

                textInFile.addAll(io.readFileAsResource("resources/1213/"
                                + MainClass.AWAY_TEAM + ".txt"));
        } catch (Exception e) {}
        ;

        for (String s : textInFile) {
                if (!(s.startsWith(MainClass.AWAY_TEAM))) {
                        game = Game.createGameFromString(s);
                        awayTeamTotalGamesAsVisitor.add(game);
                }
        }
        return awayTeamTotalGamesAsVisitor;
}


public void generateTestFile() {

        double[][] test = new double[100][7];

        // for normalization
        int maxHomeTeamScored = 1;
        int minHomeTeamScored = 200;

        int maxHomeTeamReceived = 1;
        int minHomeTeamReceived = 200;

        // for computing the values from test for home team
        int totalNumberOfGamesHomeTeam = 0;
        int pointsHomeTeamScored = 0;
        int pointsHomeTeamReceived = 0;
        int greatertThanMaxGamesHomeTeam = 0;
```

```
		int maxAwayTeamScored = 10;
		int minAwayTeamScored = 200;

		int maxAwayTeamReceived = 1;
		int minAwayTeamReceived = 200;

		int totalNumberOfGamesAwayTeam = 0;
		int pointsAwayTeamScored = 0;
		int pointsAwayTeamReceived = 0;
		int greatertThanMaxGamesAwayTeam = 0;

		int entryNo = 0;

		ArrayList<Game> homeGames = new ArrayList<Game>();
		ArrayList<Game> awayGames = new ArrayList<Game>();

		homeGames = createListHomeTeamTotalGamesAsHost();
		awayGames = createListAwayTeamTotalGamesAway();

		for (Game g : homeGames) {
			if (g.getHomeTeamScore() + g.getAwayTeamScore() >
absolutMaxScore) {
				absolutMaxScore = g.getHomeTeamScore() +
g.getAwayTeamScore();
			}
		}
		for (Game g : awayGames) {
			if (g.getHomeTeamScore() + g.getAwayTeamScore() >
absolutMaxScore) {
				absolutMaxScore = g.getHomeTeamScore() +
g.getAwayTeamScore();
			}
		}

		for (int i = 0; i < 100; i++) {
			for (int j = 0; j < 7; j++) {
				test[i][j] = 0.0;
```

```
                }

            }

            for (Game g : homeGames) {
                if (g.getAwayTeam().equals(MainClass.AWAY_TEAM)) {
                    if (totalNumberOfGamesHomeTeam != 0) {

                        double biggestDifferenceHomeTeamScored =
maxHomeTeamScored
                                - minHomeTeamScored;

                        double averageHomeTeamScoredUntilNow =
(double) pointsHomeTeamScored
                                / (double)
totalNumberOfGamesHomeTeam;

                        double averageDifferenceHomeTeamScored =
averageHomeTeamScoredUntilNow
                                - minHomeTeamScored;

                        if (biggestDifferenceHomeTeamScored == 0.0) {
                            test[entryNo][0] = 1.0;
                        } else {
                            test[entryNo][0] =
averageDifferenceHomeTeamScored
                                    /
biggestDifferenceHomeTeamScored;
                        }

                        double biggestDifferenceHomeTeamReceived =
maxHomeTeamReceived
                                - minHomeTeamReceived;
```

```
                        double averageHomeTeamReceivedUntilNow =
(double) pointsHomeTeamReceived
                                        / (double)
totalNumberOfGamesHomeTeam;

                        double averageDifferenceHomeTeamReceived =
averageHomeTeamReceivedUntilNow
                                        - minHomeTeamReceived;

                        if (biggestDifferenceHomeTeamReceived == 0.0) {
                                test[entryNo][1] = 1.0;
                        } else {
                                test[entryNo][1] =
averageDifferenceHomeTeamReceived
                                                /
biggestDifferenceHomeTeamReceived;
                        }

                        test[entryNo][4] = (double)
greatertThanMaxGamesHomeTeam
                                        / (double)
totalNumberOfGamesHomeTeam;

                        test[entryNo][6] = (double) (g.getHomeTeamScore()
+ g
                                        .getAwayTeamScore()) / (double)
(absolutMaxScore);

                }

                else {
                        test[entryNo][0] = 0;
                        test[entryNo][1] = 0;
                        test[entryNo][4] = 0;
                }

                entryNo++;
```

```
                }
                if (g.getHomeTeamScore() > maxHomeTeamScored) {
                        maxHomeTeamScored = g.getHomeTeamScore();
                }
                if (g.getHomeTeamScore() < minHomeTeamScored) {
                        minHomeTeamScored = g.getHomeTeamScore();
                }
                if (g.getAwayTeamScore() > maxHomeTeamReceived) {
                        maxHomeTeamReceived = g.getAwayTeamScore();
                }
                if (g.getAwayTeamScore() < minHomeTeamReceived) {
                        minHomeTeamReceived = g.getAwayTeamScore();
                }

                totalNumberOfGamesHomeTeam++;
                pointsHomeTeamScored = pointsHomeTeamScored +
g.getHomeTeamScore();
                pointsHomeTeamReceived = pointsHomeTeamReceived
                                + g.getAwayTeamScore();
                if (g.getHomeTeamScore() + g.getAwayTeamScore() > treshold) {
                        greatertThanMaxGamesHomeTeam++;
                }

        }
        entryNo = 0;
        for (Game g : awayGames) {
                if (g.getHomeTeam().equals(MainClass.HOME_TEAM)) {
                        if (totalNumberOfGamesAwayTeam != 0) {


                                double biggestDifferenceAwayTeamScored =
maxAwayTeamScored
                                                - minAwayTeamScored;
```

```
                        double averageAwayTeamScoredUntilNow =
(double) pointsAwayTeamScored
                                / (double)
totalNumberOfGamesAwayTeam;

                        double averageDifferenceAwayTeamScored =
averageAwayTeamScoredUntilNow
                                - minAwayTeamScored;

                        if (biggestDifferenceAwayTeamScored == 0.0) {
                                test[entryNo][2] = 1.0;
                        } else {
                                test[entryNo][2] =
averageDifferenceAwayTeamScored
                                        /
biggestDifferenceAwayTeamScored;
                        }

                        double biggestDifferenceAwayTeamReceived =
maxAwayTeamReceived
                                - minAwayTeamReceived;

                        double averageAwayTeamReceivedUntilNow =
(double) pointsAwayTeamReceived
                                / (double)
totalNumberOfGamesAwayTeam;

                        double averageDifferenceAwayTeamReceived =
averageAwayTeamReceivedUntilNow
                                - minAwayTeamReceived;

                        if (biggestDifferenceAwayTeamReceived == 0.0) {
                                test[entryNo][3] = 1.0;
                        } else {
                                test[entryNo][3] =
averageDifferenceAwayTeamReceived
```

```
                                                                    /
biggestDifferenceAwayTeamReceived;
                                        }
                                        test[entryNo][5] = (double)
greatertThanMaxGamesAwayTeam
                                                                / (double)
totalNumberOfGamesAwayTeam;

                                } else {
                                        test[entryNo][2] = 0;
                                        test[entryNo][3] = 0;
                                        test[entryNo][5] = 0;
                                }

                                entryNo++;

                        }

                        if (g.getAwayTeamScore() > maxAwayTeamScored) {
                                maxAwayTeamScored = g.getAwayTeamScore();
                        }
                        if (g.getAwayTeamScore() < minAwayTeamScored) {
                                minAwayTeamScored = g.getAwayTeamScore();
                        }
                        if (g.getHomeTeamScore() > maxAwayTeamReceived) {
                                maxAwayTeamReceived = g.getHomeTeamScore();
                        }
                        if (g.getHomeTeamScore() < minAwayTeamReceived) {
                                minAwayTeamReceived = g.getHomeTeamScore();
                        }

                        // increment everything with new info
                        totalNumberOfGamesAwayTeam++;
                        pointsAwayTeamScored = pointsAwayTeamScored +
g.getAwayTeamScore();
                        pointsAwayTeamReceived = pointsAwayTeamReceived
                                        + g.getHomeTeamScore();
```

```
                if (g.getHomeTeamScore() + g.getAwayTeamScore() > treshold) {
                        greatertThanMaxGamesAwayTeam++;
                }

        }

        int lastValidEntry = 0;
        boolean validEntry = true;
        while (validEntry) {
                for (int i = 0; i < 100; i++) {
                        for (int j = 0; j < 7; j++) {
                                if (test[i][j] != 0.0) {
                                        lastValidEntry = i;
                                        validEntry = true;
                                        break;
                                }
                                validEntry = false;

                        }

                }
        }
        StringBuffer sPP = new StringBuffer();

        for (int i = 0; i < lastValidEntry; i++) {
                boolean oneInvalidEntry = false;
                for (int j = 0; j < 7; j++) {

                        if (test[i][j] == 0.0) {
                                oneInvalidEntry = true;
                        }
                }
                String oneEntry = "";
                if (!oneInvalidEntry) {
                        for (int j = 0; j < 7; j++) {
                                oneEntry = oneEntry + String.valueOf(test[i][j]) + "
";
```

```
                        }
                }
                oneEntry = oneEntry + "\n";
                if (!(oneEntry.equals("\n"))) {
                        sPP.append(oneEntry);
                }
        }

        InputOutput.writeToFile("training.txt", sPP);

        StringBuffer sPP1 = new StringBuffer();

        String oneEntry = "";
        for (int j = 0; j < 6; j++) {
                oneEntry = oneEntry + String.valueOf(test[lastValidEntry][j]) + " ";
        }
        oneEntry = oneEntry
                        + String.valueOf((int) Math.ceil(test[lastValidEntry][6]
                                        * absolutMaxScore));
        oneEntry = oneEntry + "\n";
        sPP1.append(oneEntry);

        InputOutput.writeToFile("test.txt", sPP1);
    }
}
```

Κλάση CalculateScore

```
import java.io.BufferedReader;
```

```
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;


public class CalculateScore {
	public static int numHL1N;
	public static int numHL2N;
	public static int numInputNeurons;
	public static int numOutNeurons;
	public static double learningRate;
	public static double momentum;
	public static int maxIterations;
	public static String trainFile = new String();
	public static String testFile = new String();
	public static ArrayList<Tdata> trainData = new ArrayList<Tdata>();
	public static ArrayList<Tdata> testData = new ArrayList<Tdata>();

	private ArrayList<InputLayer> inlayer = new ArrayList<InputLayer>();
	private ArrayList<HiddenLayer> hiddenLayer1 = new ArrayList<HiddenLayer>();
	private ArrayList<HiddenLayer> hiddenLayer2 = new ArrayList<HiddenLayer>();
	private ArrayList<OutputLayer> outlayer = new ArrayList<OutputLayer>();
	private double desiredOut = 0;
	private double tError = 0;
	private ArrayList<Double> trainError = new ArrayList<Double>();
	public void calculate() {
		System.out.println("Training starts.Please wait..");
		this.readFromFile("parameters.txt");
		this.makeInputLayer();
		if (CalculateScore.numHL1N != 0) {
			this.makeHiddenLayer();
		}
		this.makeOutputLayer();
		this.readFromFile(CalculateScore.trainFile, true);
```

```
		for (int i = 0; i < CalculateScore.maxIterations; i++) {
			this.calc(CalculateScore.trainData, true);
			trainError.add(tError);

		}

		PrintInFile pf = new PrintInFile();
		pf.printResults("results.txt", trainError);
		pf.printWeights("weights.txt", inlayer, hiddenLayer1, hiddenLayer2);
		System.out.println("End of the training..");

	}


	public void readFromFile(String filename) {
		ReadFromFile rf = new ReadFromFile();
		rf.readFromFile(filename);
	}


	public void readFromFile(String filename, boolean f) {
		ReadFromFile rf = new ReadFromFile();
		rf.readFromFile(filename, f);
	}

	public void makeInputLayer() {
		ArrayList<Double> w = new ArrayList<Double>();
		ArrayList<Double> temp = null;
		if (CalculateScore.numHL1N != 0) {
			for (int i = 0; i <= (CalculateScore.numInputNeurons); i++) {
				for (int j = 0; j <= CalculateScore.numHL1N; j++) {
					w.add(randomWeight());
					temp = new ArrayList<Double>(w);
				}
				InputLayer hl = new InputLayer(i, temp);
				inlayer.add(hl);
				w.clear();
```

```
                }
        } else {
                for (int i = 0; i <= (CalculateScore.numInputNeurons); i++) {
                        for (int j = 0; j < CalculateScore.numOutNeurons; j++) {
                                w.add(randomWeight());
                                temp = new ArrayList<Double>(w);
                        }
                        InputLayer hl = new InputLayer(i, temp);
                        inlayer.add(hl);
                        w.clear();
                }
        }
}


public double randomWeight() {

        Random r = new Random();
        double w = (r.nextDouble() * 2) - 1;

        return w;

}


public void makeHiddenLayer() {
        ArrayList<Double> w = new ArrayList<Double>();
        ArrayList<Double> tempW = null;
        for (int i = 0; i <= CalculateScore.numHL1N; i++) {
                if (CalculateScore.numHL2N != 0) {
                        for (int j = 0; j <= CalculateScore.numHL2N; j++) {
                                w.add(randomWeight());
                                tempW = new ArrayList<Double>(w);
                        }
                } else {
                        for (int j = 0; j < CalculateScore.numOutNeurons; j++) {
                                w.add(randomWeight());
```

```
                        tempW = new ArrayList<Double>(w);
                    }
                }
                HiddenLayer hl = new HiddenLayer(i, 1, tempW);
                hiddenLayer1.add(hl);
                w.clear();
            }
            if (CalculateScore.numHL2N != 0) {
                for (int i = 0; i <= CalculateScore.numHL2N; i++) {
                    for (int j = 0; j < CalculateScore.numOutNeurons; j++) {
                        w.add(randomWeight());
                        tempW = new ArrayList<Double>(w);
                    }
                    HiddenLayer hl = new HiddenLayer(i, 1, tempW);
                    hiddenLayer2.add(hl);
                    w.clear();
                }
            }

    }


    public void makeOutputLayer() {
            for (int i = 0; i < CalculateScore.numOutNeurons; i++) {
                OutputLayer ol = new OutputLayer(i);
                outlayer.add(ol);

            }
    }


    public void calc(ArrayList<Tdata> tData, boolean f) {
            tError = 0;
            for (int pno = 0; pno < tData.size(); pno++) {
                inlayer.get(0).setInput(tData.get(pno).input1);
                inlayer.get(1).setInput(tData.get(pno).input2);
                inlayer.get(2).setInput(tData.get(pno).input3);
```

```
                inlayer.get(3).setInput(tData.get(pno).input4);
                inlayer.get(4).setInput(tData.get(pno).input5);
                inlayer.get(5).setInput(tData.get(pno).input6);



                desiredOut = tData.get(pno).output;
                this.calcOutput();
                if (f) {
                        this.calcErrorWeights();
                }
                this.calcTerror();
        }
        tError = tError / tData.size();
}



public void calcOutput() {
        FeedForward ff = new FeedForward();
        if (CalculateScore.numHL1N != 0) {
                ff.calculateSum_l1(inlayer, hiddenLayer1);
                ff.sigmoidHidden(hiddenLayer1);
                if (CalculateScore.numHL2N != 0) {
                        ff.calculateSum_hidden(hiddenLayer1, hiddenLayer2);
                        ff.sigmoidHidden(hiddenLayer2);
                        ff.calculateSum_out(hiddenLayer2, outlayer);
                } else {
                        ff.calculateSum_out(hiddenLayer1, outlayer);
                }
        } else {
                ff.calculateSum_out2(inlayer, outlayer);
        }
        ff.sigmoidOut(outlayer);
}
```

```
	public void calcErrorWeights() {
		BackPropagation bp = new BackPropagation();
		for (int i = 0; i < outlayer.size(); i++) {
			bp.calculateErrorOl(outlayer.get(i), desiredOut);
			if (CalculateScore.numHL2N != 0) {
				bp.calculateNewWeightsHFinal(hiddenLayer2,
outlayer.get(i));
			} else if (CalculateScore.numHL1N != 0) {
				bp.calculateNewWeightsHFinal(hiddenLayer1,
outlayer.get(i));
			} else {
				bp.calculateNewWeightsIO(inlayer, outlayer.get(i));
			}

		}
		if (CalculateScore.numHL2N != 0) {
			for (int i = 0; i < hiddenLayer2.size(); i++) {
				bp.calculateErrorHFinal(hiddenLayer2.get(i), outlayer);
				bp.calculateNewWeightsHl(hiddenLayer2.get(i),
hiddenLayer1);
			}
			for (int i = 0; i < hiddenLayer1.size(); i++) {
				bp.calculateErrorHl(hiddenLayer1.get(i), hiddenLayer2);
				bp.calculateNewWeightsIl(hiddenLayer1.get(i), inlayer);
			}			} else if (CalculateScore.numHL1N != 0) {
			for (int i = 0; i < hiddenLayer1.size(); i++) {
				bp.calculateErrorHFinal(hiddenLayer1.get(i), outlayer);
				bp.calculateNewWeightsIl(hiddenLayer1.get(i), inlayer);
			}
		}
	}



	public void calcTerror() {
```

```
            tError += Math.abs((desiredOut - outlayer.get(0).output));

    }


    public int[] calculateOneResult(){
            int[] results = new int[2];
            BufferedReader in = null;
            testFile = "test.txt";
            try {
                    in = new BufferedReader(new FileReader(testFile));
            } catch (FileNotFoundException e) {
                    e.printStackTrace();
            }

            String s = "";
            try {
                    s = in.readLine();
                    in.close();
            } catch (IOException e) {
                    e.printStackTrace();
            }


            int actualResult =0 ;
            int space = s.indexOf(" ");
            inlayer.get(0).setInput(Double.valueOf(s.substring(0, space)));
            s = s.substring(space + 1);

            space = s.indexOf(" ");
            inlayer.get(1).setInput(Double.valueOf(s.substring(0, space)));
            s = s.substring(space + 1);


            space = s.indexOf(" ");
            inlayer.get(2).setInput(Double.valueOf(s.substring(0, space)));
            s = s.substring(space + 1);
```

```
                space = s.indexOf(" ");
                inlayer.get(3).setInput(Double.valueOf(s.substring(0, space)));
                s = s.substring(space + 1);


                space = s.indexOf(" ");
                inlayer.get(4).setInput(Double.valueOf(s.substring(0, space)));


                s = s.substring(space + 1);
                space = s.indexOf(" ");
                inlayer.get(5).setInput(Double.valueOf(s.substring(0, space)));


                s = s.substring(space + 1);

                actualResult = Integer.valueOf(s);
        }

                this.calcOutput();

                int predictedResult = (int)Math.ceil(outlayer.get(0).output *
GenerateTest.absolutMaxScore);

                results[0] = predictedResult;
                results[1] = actualResult;

        return results;

        }
}
```

Κλάση MainClass

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class MainClass {
	public static String HOME_TEAM;
	public static String AWAY_TEAM;
	public static InputOutput io = new InputOutput();
	public static GenerateTest gt = new GenerateTest();
	public static CalculateScore c = new CalculateScore();
	public static double treshold = 200;

	public static void readInputFor2() {
		boolean valid = false;
		io = new InputOutput();
		ArrayList<String> teams = io
				.readFileAsResource("resources/0405/TEAMS.txt");
		teams.add("OKLAHOMA");

		System.out
				.println("Please enter the name of the home team, with
capital letters: ");

		do {

			HOME_TEAM = new Scanner(System.in).nextLine();
			if (teams.contains(HOME_TEAM)) {
				valid = true;
			} else {
				System.err
						.println("No information about this team.
Please try again! ");
				valid = false;
			}
```

```
                } while (valid == false);
                System.out
                                .println("Please enter the name of the away team, with capital
letters: ");

                valid = false;
                do {

                        AWAY_TEAM = new Scanner(System.in).nextLine();
                        if (teams.contains(AWAY_TEAM)) {
                                valid = true;
                        } else {
                                System.err
                                                .println("No information about this team.
Please try again! ");
                                valid = false;
                        }

                } while (valid == false);
        }

        public static void main(String[] args) {
                System.out.println("This program predicts the game results");
                System.out.println("Please select what you want to do: 1 for predicting a
specific game result; 2 for testing for the entire data set.");
                int choice = 0;
                try{
                        choice = new Scanner(System.in).nextInt();
                }
                catch(Exception e){};
                while ((choice != 1) && (choice != 2)) {
                        System.out
                                        .println("You gave wrong input data.The choice
must be 1 or 2..");
                        System.out.print("Please make your choice:");
                        choice = new Scanner(System.in).nextInt();
                }
```

```
                if (choice == 1) {
                        readInputFor2();
                        System.out.println("generate for " + HOME_TEAM + " " +
AWAY_TEAM);
                        gt = new GenerateTest();
                        gt.generateTestFile();
                        c = new CalculateScore();
                        c.calculate();
                        choice = 0;
                        while (true) {
                                System.out.println("\n");
                                System.out
                                                .println("If you want to continue(to test the
results) press 1:");
                                System.out.println("If you want to exit press 2:");
                                System.out.print("Please make your choice:");
                                try{
                                        choice = new Scanner(System.in).nextInt();
                                }
                                catch(Exception e){};
                                while ((choice != 1) && (choice != 2)) {
                                        System.out
                                                        .println("You gave wrong input
data.The choice must be 1 or 2..");
                                        System.out.print("Please make your choice:");
                                        choice = new Scanner(System.in).nextInt();
                                }
                                if (choice == 2) {
                                        System.out.print("Exiting..");
                                        System.exit(-1);
                                }
                                int[] res = c.calculateOneResult();
                                System.out.println("The output is:" + res[0]);
                                System.out.println("The correct result is:" + res[1]);
                        }

                }
```

```
if (choice == 2) {
        InputOutput io = new InputOutput();
        ArrayList<String> teams = io
                        .readFileAsResource("resources/0405/TEAMS.txt");
        teams.add("OKLAHOMA");

        int correctRatio = 0;
        int totalGames = 0;
        try {

            FileWriter fw = new FileWriter("OVERALL.csv");
          PrintWriter pw = new PrintWriter(fw);
          pw.println("Home Team,Away Team,Prediction,Actual,Correct");

          for (String host : teams){
                    for (String away : teams){
                        if (!(host.equals(away))){
                            totalGames ++;

                            HOME_TEAM = host;
                            AWAY_TEAM = away;
                            System.out.println("generate for " +
HOME_TEAM + " " + AWAY_TEAM);
                            gt = new GenerateTest();
                            gt.generateTestFile();
                            c = new CalculateScore();
                            c.calculate();
                            int[] res = c.calculateOneResult();
                            boolean correct = false;

                            if( ((res[0] >= treshold) && (res[1]
>=treshold)) || ((res[0] < treshold) && (res[1] < treshold)) ){
                                correct = true;
                                correctRatio ++;
                            }
```

```
                                                    pw.println(HOME_TEAM + "," +
AWAY_TEAM + "," + res[0] +","+ res[1] + "," + String.valueOf(correct));


                                                }
                                            }
                                        }

                        pw.flush();

                        pw.close();

                        fw.close();

                            }
                            catch (IOException e) {
                                    System.err.println("Unable to write to file");
                                    System.exit(-1);
                            }


                    System.out.println("Finished.");
                            System.out.println("The percentage of correct predictions is: "+
String.valueOf((double)correctRatio/ (double) totalGames));
                            System.out.println("Detailed score results can be found in
OVERALL.csv");
                            }
            }

}
```

ΠΑΡΑΡΤΗΜΑ Β – Υλοποίηση RBF

Κλάση TestTraining.Java

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;


public class TestTraining
{
	private int numberOfLines=0;
	ArrayList<ArrayList<String>> dataset = new
ArrayList<ArrayList<String>>();
	private FileInputStream fstream = null;

	TestTraining(String nameOfFile, int noOfColumns) throws IOException
	{

		fstream = new FileInputStream(nameOfFile);

		DataInputStream in = new DataInputStream(fstream);
		BufferedReader br = new BufferedReader(new
InputStreamReader(in));
		String strLine;

		while ((strLine = br.readLine()) != null)
		{
			++numberOfLines;

			String[] tempArray = strLine.split(",");
			ArrayList<String> tempArrayList = new
ArrayList<String>();

			for(int i=0; i<tempArray.length; i++)
			{
				tempArrayList.add(tempArray[i]);
			}


			dataset.add(tempArrayList);
		}
		in.close();
	}

	public ArrayList<ArrayList<String>> getTheArray()
	{
		return dataset;
	}
	public ArrayList<String> getColumn(int col)
	{
		for(int i=0; i<dataset.size(); i++)
```

```
		{
			return dataset.get(col);
		}
		return null;

	}

	public int getNumberOfColumns()
	{
		return dataset.size();
	}

	public int getNumberOfLines()
	{
		return numberOfLines;
	}


}
```

Κλάση Parameters.Java

```
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;


public class Parameters
{
	String noOfHiddenNeurons;
	String noOfInputValues;
	String noOfOutputNeurons;
	String learningRateWeight;
	String learningRateCenter;
	String learningRateSigma;
	String sigmas;
	String iterations;
	String centersFile;
	String trainFile;
	String testFile;

	Parameters() throws IOException
	{
		ArrayList<String> arrayList = new ArrayList<String>();
		FileInputStream fstream = null;
		try
		{
			fstream = new FileInputStream("parameters.txt");
		} catch (FileNotFoundException e)
		{
			e.printStackTrace();
		}
```

```
            DataInputStream in = new DataInputStream(fstream);
            BufferedReader br = new BufferedReader(new
InputStreamReader(in));
            String strLine;

            while ((strLine = br.readLine()) != null)
            {

                  String[] tempArray = strLine.split(" ");
                  arrayList.add(tempArray[1]);
            }
            in.close();
            noOfHiddenNeurons=arrayList.get(0);
            noOfInputValues=arrayList.get(1);
            noOfOutputNeurons=arrayList.get(2);
            learningRateWeight=arrayList.get(3);
            learningRateCenter=arrayList.get(4);
            learningRateSigma=arrayList.get(5);
            sigmas=arrayList.get(6);
            iterations=arrayList.get(7);
            centersFile=arrayList.get(8);
            trainFile=arrayList.get(9);
            testFile=arrayList.get(10);
      }

      String getString()
      {
            return noOfHiddenNeurons+" "+noOfInputValues+"
"+noOfOutputNeurons+" "+learningRateWeight+" "+learningRateCenter+"
"+learningRateSigma+" "+sigmas+" "+iterations+" "+centersFile+"
"+trainFile+" "+testFile;
      }

}
```

Κλάση InputOutput.Java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.ArrayList;



public class InputOutput {
      public static void writeToFile(String fileName, StringBuffer
outputEntries) {
            File myFile = new File(fileName);
            try {
                  myFile.createNewFile();
```

```
			} catch (IOException e1) {
				e1.printStackTrace();
			}
			FileWriter output;
			try {
				output = new FileWriter(fileName);
				output.write(outputEntries.toString());

				output.flush();
				output.close();
			} catch (IOException e) {
			}

		}


		public ArrayList<String> readFileAsResource(String fileName) {

			ArrayList<String> result = new ArrayList<String>();

			InputStream is = getClass().getResourceAsStream(fileName);
			InputStreamReader isr = new InputStreamReader(is);
			BufferedReader br = null;
			String line = "";
			try {
				br = new BufferedReader(isr);
				while ((line = br.readLine()) != null) {
					result.add(line);
				}

			} catch (FileNotFoundException e) {
				e.printStackTrace();
			} catch (IOException e) {
				e.printStackTrace();
			} finally {
				if (br != null) {
					try {
						br.close();
					} catch (IOException e) {
						e.printStackTrace();
					}
				}
			}
			return result;
		}

	}
```

Κλάση InputNeuron.Java

```
import java.util.ArrayList;

public class InputNeuron
{
	private ArrayList<Double> inputs = new ArrayList<Double>();
```

```
	private Double value=0.0;

	public void setInputs(Double value)
	{
		this.value=value;
	}

	public Double getValue()
	{
		return value;
	}


}
```

Κλάση Neuron.Java

```
import java.util.ArrayList;

public class Neuron
{
	ArrayList<Double> weights = new ArrayList<Double>();

	ArrayList<Double> neuronInput=new ArrayList<Double>();

	ArrayList<Double> previousWeights=new ArrayList<Double>();

	ArrayList<Double> R=new ArrayList<Double>();

	private double output;
	private double bias;
	private double sigma;
	private double basicWeight;
	private double prevWeight;


	Neuron(int numOfInputs, ArrayList<Double> centers)
	{
		for(int i=0; i<numOfInputs; ++i)
		{
			weights.add(random(-1,1));
		}

		R=centers;
		sigma=random(0,1);
		bias=random(0,1);
		basicWeight=random(0,1);
	}


	public void updateNetworkParameters(Double
learningRateOfWeight,Double learningRateOfS,Double learningRateOfR
,ArrayList<Double> inputArrayList,Double error)
	{
		prevWeight=basicWeight;
```

```
	basicWeight+=learningRateOfWeight*computeGaussian(inputArrayList)*er
ror;
		sigma
+=learningRateOfS*error*prevWeight*computeGaussian(inputArrayList)*computeE
uclidianDistance(inputArrayList)/Math.pow(sigma, 3);
		for(int i=0; i<R.size(); ++i)
		{
			Double currentR = R.get(i);

	currentR+=learningRateOfR*error*prevWeight*euclidian(inputArrayList)
*computeGaussian(inputArrayList)/Math.pow(sigma, 2);
		}
	}

	public Double getOutput()
	{
		return basicWeight*output;
	}


	public void computeOutput(ArrayList<Double> inputArrayList)
	{
		output=computeGaussian(inputArrayList);
	}


	private Double computeGaussian(ArrayList<Double> inputArrayList)
	{
		double output = Math.exp(-
(computeEuclidianDistance(inputArrayList ) / (2*Math.pow(sigma, 2))));
		return output;
	}


	private Double computeEuclidianDistance(ArrayList<Double>
inputArrayList)
	{
		Double distance=0.0;
		Double newDistance=0.0;
		for(int i=0; i<R.size(); ++i)
		{
			distance =inputArrayList.get(i)-R.get(i);
			distance=Math.pow(distance, 2);
			newDistance+=distance;
		}

		return newDistance;
	}




	private Double euclidian(ArrayList<Double> inputArrayList)
	{
		Double distance=0.0;
		Double newDistance=0.0;
		for(int i=0; i<R.size(); ++i)
```

```
		{
			distance =inputArrayList.get(i)-R.get(i);
			newDistance+=distance;
		}

		return newDistance;
	}


	public void setOutput(double output)
	{
		this.output = output;
	}

	public void setBias(double bias)
	{
		this.bias = bias;
	}

	public double getBias()
	{
		return bias;
	}

	public void setsGreek(double sigma)
	{
		this.sigma = sigma;
	}

	public double getsGreek()
	{
		return sigma;
	}
	public double random( double min, double max )
	{
	  double diff = max - min;
	  return min + Math.random( ) * diff;
	}
}
```

Κλάση Layer.Java

```
import java.util.ArrayList;

public class Layer
{
	ArrayList<Neuron> neurons = new ArrayList<Neuron>();
	ArrayList<InputNeuron> inputNeurons = new ArrayList<InputNeuron>();
	private int sizeOfLayer;

	public int size()
	{
		return sizeOfLayer;
	}
}
```

Κλάση GenerateTest.java

```
import java.io.IOException;
import java.util.ArrayList;

public class GenerateTest {

	static int absolutMaxScore = 0;
	public static double treshold = 200;



	public ArrayList<Game> createListHomeTeamTotalGamesAsHost() {
		ArrayList<Game> homeTeamTotalGamesAsHost = new
ArrayList<Game>();
		Game game = new Game();
		InputOutput io = new InputOutput();
		ArrayList<String> textInFile = new ArrayList<String>();

		try {
			textInFile = io.readFileAsResource("resources/0405/"
					+ MainClass.HOME_TEAM + ".txt");
		} catch (Exception e) {
		}
		;

		try {

	textInFile.addAll(io.readFileAsResource("resources/0506/"
					+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;
		try {

	textInFile.addAll(io.readFileAsResource("resources/0607/"
					+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;
		try {


	textInFile.addAll(io.readFileAsResource("resources/0708/"
					+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;
```

```
		try {


	textInFile.addAll(io.readFileAsResource("resources/0809/"
				+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;
		try {


	textInFile.addAll(io.readFileAsResource("resources/0910/"
				+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;
		try {


	textInFile.addAll(io.readFileAsResource("resources/1011/"
				+ MainClass.HOME_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;

		for (String s : textInFile) {
			if (s.startsWith(MainClass.HOME_TEAM)) {
				game = Game.createGameFromString(s);
				homeTeamTotalGamesAsHost.add(game);
			}
		}
		return homeTeamTotalGamesAsHost;
	}


	public ArrayList<Game> createListAwayTeamTotalGamesAway() {
		ArrayList<Game> awayTeamTotalGamesAsVisitor = new
ArrayList<Game>();
		Game game = new Game();
		InputOutput io = new InputOutput();
		ArrayList<String> textInFile = new ArrayList<String>();

		try {
			textInFile = io.readFileAsResource("resources/0405/"
				+ MainClass.AWAY_TEAM + ".txt");
		} catch (Exception e) {
		}
		;

		try {


	textInFile.addAll(io.readFileAsResource("resources/0506/"
				+ MainClass.AWAY_TEAM + ".txt"));
		} catch (Exception e) {
		}
		;

		try {
```

```
textInFile.addAll(io.readFileAsResource("resources/0607/"
                        + MainClass.AWAY_TEAM + ".txt"));
      } catch (Exception e) {
      }
      ;

      try {


textInFile.addAll(io.readFileAsResource("resources/0708/"
                        + MainClass.AWAY_TEAM + ".txt"));
      } catch (Exception e) {
      }
      ;

      try {


textInFile.addAll(io.readFileAsResource("resources/0809/"
                        + MainClass.AWAY_TEAM + ".txt"));
      } catch (Exception e) {
      }
      ;

      try {


textInFile.addAll(io.readFileAsResource("resources/0910/"
                        + MainClass.AWAY_TEAM + ".txt"));
      } catch (Exception e) {
      }
      ;

      try {


textInFile.addAll(io.readFileAsResource("resources/1011/"
                        + MainClass.AWAY_TEAM + ".txt"));
      } catch (Exception e) {
      }
      ;

      for (String s : textInFile) {
            if (!(s.startsWith(MainClass.AWAY_TEAM))) {
                  game = Game.createGameFromString(s);
                  awayTeamTotalGamesAsVisitor.add(game);
            }
      }
      return awayTeamTotalGamesAsVisitor;
}


public void generateTestFile() throws IOException {

      double[][] test = new double[100][11];

      int maxHomeTeamScored = 1;
```

```
		int minHomeTeamScored = 200;

		int maxHomeTeamReceived = 1;
		int minHomeTeamReceived = 200;

		int totalNumberOfGamesHomeTeam = 0;
		int pointsHomeTeamScored = 0;
		int pointsHomeTeamReceived = 0;
		int greatertThanMaxGamesHomeTeam = 0;

		int maxAwayTeamScored = 10;
		int minAwayTeamScored = 200;

		int maxAwayTeamReceived = 1;
		int minAwayTeamReceived = 200;

		int totalNumberOfGamesAwayTeam = 0;
		int pointsAwayTeamScored = 0;
		int pointsAwayTeamReceived = 0;
		int greatertThanMaxGamesAwayTeam = 0;

		int entryNo = 0;

		ArrayList<Game> homeGames = new ArrayList<Game>();
		ArrayList<Game> awayGames = new ArrayList<Game>();

		homeGames = createListHomeTeamTotalGamesAsHost();
		awayGames = createListAwayTeamTotalGamesAway();

		absolutMaxScore = 0;
		for (Game g : homeGames) {
			if (g.getHomeTeamScore() + g.getAwayTeamScore() >
absolutMaxScore) {
				absolutMaxScore = g.getHomeTeamScore() +
g.getAwayTeamScore();
			}
		}
		for (Game g : awayGames) {
			if (g.getHomeTeamScore() + g.getAwayTeamScore() >
absolutMaxScore) {
				absolutMaxScore = g.getHomeTeamScore() +
g.getAwayTeamScore();
			}
		}

		for (int i = 0; i < 100; i++) {
			for (int j = 0; j < 11; j++) {
				test[i][j] = 0.0;
			}

		}

		int indexHome = -1;
		for (Game g : homeGames) {
			indexHome ++;
			if (g.getAwayTeam().equals(MainClass.AWAY_TEAM)) {
				if (totalNumberOfGamesHomeTeam != 0) {
```

```
                                double biggestDifferenceHomeTeamScored =
maxHomeTeamScored
                                            - minHomeTeamScored;


                                double averageHomeTeamScoredUntilNow =
(double) pointsHomeTeamScored
                                            / (double)
totalNumberOfGamesHomeTeam;

                                                                    double
averageDifferenceHomeTeamScored = averageHomeTeamScoredUntilNow
                                            - minHomeTeamScored;

                                if (biggestDifferenceHomeTeamScored ==
0.0) {
                                      test[entryNo][0] = 1.0;
                                } else {
                                      test[entryNo][0] =
averageDifferenceHomeTeamScored
                                                  /
biggestDifferenceHomeTeamScored;
                                }

                                double biggestDifferenceHomeTeamReceived =
maxHomeTeamReceived
                                            - minHomeTeamReceived;

                                double averageHomeTeamReceivedUntilNow =
(double) pointsHomeTeamReceived
                                            / (double)
totalNumberOfGamesHomeTeam;

                                double averageDifferenceHomeTeamReceived =
averageHomeTeamReceivedUntilNow
                                            - minHomeTeamReceived;

                                if (biggestDifferenceHomeTeamReceived ==
0.0) {
                                      test[entryNo][1] = 1.0;
                                } else {
                                      test[entryNo][1] =
averageDifferenceHomeTeamReceived
                                                  /
biggestDifferenceHomeTeamReceived;
                                }

                                test[entryNo][4] = (double)
greatertThanMaxGamesHomeTeam
                                            / (double)
totalNumberOfGamesHomeTeam;

                                test[entryNo][10] = (double)
(g.getHomeTeamScore() + g
                                            .getAwayTeamScore()) /
(double) (absolutMaxScore);

                                double pointsHomeTeamScoredUntilNow6 =
0.0;
```

```
                                    double pointsHomeTeamReceivedUntilNow6 =
0.0;
                                    double averageHomeTeamScoredUntilNow6 =
0.0;
                                    double averageHomeTeamReceivedUntilNow6 =
0.0;
                                    int prevHome = 0;
                                    for (int indexHome6 =1; indexHome6 < 7;
indexHome6 ++){

                                          if(indexHome -indexHome6 > 0){
                                                prevHome ++ ;
                                                pointsHomeTeamScoredUntilNow6
= pointsHomeTeamScoredUntilNow6 + homeGames.get(indexHome -
indexHome6).homeTeamScore;

      pointsHomeTeamReceivedUntilNow6 = pointsHomeTeamReceivedUntilNow6 +
homeGames.get(indexHome -indexHome6).awayTeamScore;
                                          }

                                    }
                                    averageHomeTeamScoredUntilNow6 = (double)
pointsHomeTeamScoredUntilNow6
                                                / (double) prevHome;
                                    averageHomeTeamReceivedUntilNow6 =
(double) pointsHomeTeamReceivedUntilNow6
                                          / (double) prevHome;

                                    if (biggestDifferenceHomeTeamScored ==
0.0) {
                                          test[entryNo][6] = 1.0;
                                    } else {
                                          test[entryNo][6] =
(averageHomeTeamScoredUntilNow6 - minHomeTeamScored) /
biggestDifferenceHomeTeamScored;
                                    }
                                    if (biggestDifferenceHomeTeamReceived ==
0.0) {
                                          test[entryNo][7] = 1.0;
                                    } else {
                                          test[entryNo][7] =
(averageHomeTeamReceivedUntilNow6 - minHomeTeamReceived) /
biggestDifferenceHomeTeamReceived;
                                    }

                              }



                              else {
                                    test[entryNo][0] = 0;
                                    test[entryNo][1] = 0;
                                    test[entryNo][4] = 0;
                                    test[entryNo][6] = 0;
                                    test[entryNo][7] = 0;
                              }
```

```
                        entryNo++;

                    }
                    if (g.getHomeTeamScore() > maxHomeTeamScored) {
                        maxHomeTeamScored = g.getHomeTeamScore();
                    }
                    if (g.getHomeTeamScore() < minHomeTeamScored) {
                        minHomeTeamScored = g.getHomeTeamScore();
                    }
                    if (g.getAwayTeamScore() > maxHomeTeamReceived) {
                        maxHomeTeamReceived = g.getAwayTeamScore();
                    }
                    if (g.getAwayTeamScore() < minHomeTeamReceived) {
                        minHomeTeamReceived = g.getAwayTeamScore();
                    }

                    totalNumberOfGamesHomeTeam++;
                    pointsHomeTeamScored = pointsHomeTeamScored +
g.getHomeTeamScore();
                    pointsHomeTeamReceived = pointsHomeTeamReceived
                                + g.getAwayTeamScore();
                    if (g.getHomeTeamScore() + g.getAwayTeamScore() >
treshold) {
                        greatertThanMaxGamesHomeTeam++;
                    }

            }
            entryNo = 0;
            int indexAway = -1;
            for (Game g : awayGames) {              indexAway ++;
                    if (g.getHomeTeam().equals(MainClass.HOME_TEAM)) {
                        if (totalNumberOfGamesAwayTeam != 0) {

                                                                double
biggestDifferenceAwayTeamScored = maxAwayTeamScored
                                                - minAwayTeamScored;


                                double averageAwayTeamScoredUntilNow =
(double) pointsAwayTeamScored
                                                / (double)
totalNumberOfGamesAwayTeam;

                                                                double
averageDifferenceAwayTeamScored = averageAwayTeamScoredUntilNow
                                                - minAwayTeamScored;

                                if (biggestDifferenceAwayTeamScored ==
0.0) {
                                    test[entryNo][2] = 1.0;
                                } else {
                                    test[entryNo][2] =
averageDifferenceAwayTeamScored
                                                    /
biggestDifferenceAwayTeamScored;
                                }

                                double biggestDifferenceAwayTeamReceived =
maxAwayTeamReceived
```

```
                                        - minAwayTeamReceived;

                                double averageAwayTeamReceivedUntilNow =
(double) pointsAwayTeamReceived
                                        / (double)
totalNumberOfGamesAwayTeam;

                                double averageDifferenceAwayTeamReceived =
averageAwayTeamReceivedUntilNow
                                        - minAwayTeamReceived;

                                                                if
(biggestDifferenceAwayTeamReceived == 0.0) {
                                        test[entryNo][3] = 1.0;
                                } else {
                                        test[entryNo][3] =
averageDifferenceAwayTeamReceived
                                                        /
biggestDifferenceAwayTeamReceived;
                                }

        test[entryNo][5] = (double) greatertThanMaxGamesAwayTeam
                                        / (double)
totalNumberOfGamesAwayTeam;


                                double pointsAwayTeamScoredUntilNow6 =
0.0;
                                double pointsAwayTeamReceivedUntilNow6 =
0.0;
                                double averageAwayTeamScoredUntilNow6 =
0.0;
                                double averageAwayTeamReceivedUntilNow6 =
0.0;
                                int prevAway = 0;
                                for (int indexAway6 =1; indexAway6 < 7;
indexAway6 ++){

                                        if(indexAway -indexAway6 > 0){
                                                prevAway ++ ;
                                                pointsAwayTeamScoredUntilNow6
= pointsAwayTeamScoredUntilNow6 + awayGames.get(indexAway -
indexAway6).awayTeamScore;

        pointsAwayTeamReceivedUntilNow6 = pointsAwayTeamReceivedUntilNow6 +
awayGames.get(indexAway -indexAway6).homeTeamScore;
                                        }

                                }
                                averageAwayTeamScoredUntilNow6 = (double)
pointsAwayTeamScoredUntilNow6
                                        / (double) prevAway;
                        averageAwayTeamReceivedUntilNow6 = (double)
pointsAwayTeamReceivedUntilNow6
                                        / (double) prevAway;

                        if (biggestDifferenceAwayTeamScored == 0.0) {
                                test[entryNo][8] = 1.0;
                        } else {
```

```
                                test[entryNo][8] =
(averageAwayTeamScoredUntilNow6 - minAwayTeamScored) /
biggestDifferenceAwayTeamScored;
                          }
                          if (biggestDifferenceAwayTeamReceived == 0.0) {
                                test[entryNo][9] = 1.0;
                          } else {
                                test[entryNo][9] =
(averageAwayTeamReceivedUntilNow6 - minAwayTeamReceived) /
biggestDifferenceAwayTeamReceived;
                          }



                          } else {
                                test[entryNo][2] = 0;
                                test[entryNo][3] = 0;
                                test[entryNo][5] = 0;
                                test[entryNo][8] = 0;
                                test[entryNo][9] = 0;
                          }

                          entryNo++;

                    }

                    if (g.getAwayTeamScore() > maxAwayTeamScored) {
                          maxAwayTeamScored = g.getAwayTeamScore();
                    }
                    if (g.getAwayTeamScore() < minAwayTeamScored) {
                          minAwayTeamScored = g.getAwayTeamScore();
                    }
                    if (g.getHomeTeamScore() > maxAwayTeamReceived) {
                          maxAwayTeamReceived = g.getHomeTeamScore();
                    }
                    if (g.getHomeTeamScore() < minAwayTeamReceived) {
                          minAwayTeamReceived = g.getHomeTeamScore();
                    }

                    totalNumberOfGamesAwayTeam++;
                    pointsAwayTeamScored = pointsAwayTeamScored +
g.getAwayTeamScore();
                    pointsAwayTeamReceived = pointsAwayTeamReceived
                                + g.getHomeTeamScore();
                    if (g.getHomeTeamScore() + g.getAwayTeamScore() >
treshold) {
                          greatertThanMaxGamesAwayTeam++;
                    }

              }

              int lastValidEntry = 0;
              boolean validEntry = true;
              while (validEntry) {
                    for (int i = 0; i < 100; i++) {
                          for (int j = 0; j < 11; j++) {
                                if (test[i][j] != 0.0) {
                                      lastValidEntry = i;
                                      validEntry = true;
```

```
                                        break;
                                }
                                validEntry = false;

                        }

                }
        }


        StringBuffer sPP = new StringBuffer();

        for (int i = 0; i < lastValidEntry; i++) {
                boolean oneInvalidEntry = false;
                for (int j = 0; j < 11; j++) {

                        if (test[i][j] == 0.0) {
                                oneInvalidEntry = true;
                        }
                }
                String oneEntry = "";
                if (!oneInvalidEntry) {
                        for (int j = 0; j < 11; j++) {
                                oneEntry = oneEntry +
String.valueOf(test[i][j]) + ",";
                        }
                }
                if(oneEntry.length()!=0){
                        oneEntry = oneEntry.substring(0,
oneEntry.length()-1);
                }
                oneEntry = oneEntry + "\n";
                if (!(oneEntry.equals("\n"))) {
                        sPP.append(oneEntry);
                }
        }

        InputOutput.writeToFile("training.txt", sPP);


        StringBuffer sPP2 = new StringBuffer();
        Parameters pr = new Parameters();
        int hiddenLayerNeurons = 0;
        if(Integer.valueOf(pr.noOfHiddenNeurons) <=
Integer.valueOf(pr.noOfInputValues)){
                hiddenLayerNeurons =
Integer.valueOf(pr.noOfHiddenNeurons) +1;
        }
        else{
                hiddenLayerNeurons =
Integer.valueOf(pr.noOfInputValues) +1;
        }


        for (int i = 1; i< hiddenLayerNeurons ; i ++ ){
                String oneEntry = "";
                for (int j = 0; j < 10; j++) {
                        if(lastValidEntry-i >= 0){
```

```
                                    oneEntry = oneEntry +
String.valueOf(test[lastValidEntry-i][j]) + ",";
                              }
                              else{
                                    oneEntry = oneEntry +
String.valueOf(test[lastValidEntry-1][j]) + ",";
                              }
                        }
                        if(oneEntry.length()!=0){
                              oneEntry = oneEntry.substring(0,
oneEntry.length()-1);
                        }
                        oneEntry = oneEntry + "\n";

                        if (!(oneEntry.equals("\n"))) {
                              sPP2.append(oneEntry);
                        }
                  }

                  InputOutput.writeToFile("centersFile.txt", sPP2);


                  StringBuffer sPP1 = new StringBuffer();

                  String oneEntry = "";
                  for (int j = 0; j < 10; j++) {
                        oneEntry = oneEntry +
String.valueOf(test[lastValidEntry][j]) + ",";
                  }
                  oneEntry = oneEntry
                              + String.valueOf((int)
Math.ceil(test[lastValidEntry][10]
                                          * absolutMaxScore));
                  oneEntry = oneEntry + "\n";
                  sPP1.append(oneEntry);

                  InputOutput.writeToFile("test.txt", sPP1);
            }
}
```

Κλάση Game.Java

```
public class Game {
      public Game() {
```

```
        super();
    }
    public Game(String homeTeam, String awayTeam, int homeTeamScore,
                int awayTeamScore) {
        super();
        this.homeTeam = homeTeam;
        this.awayTeam = awayTeam;
        this.homeTeamScore = homeTeamScore;
        this.awayTeamScore = awayTeamScore;
    }
    public String getHomeTeam() {
        return homeTeam;
    }
    public void setHomeTeam(String homeTeam) {
        this.homeTeam = homeTeam;
    }
    public String getAwayTeam() {
        return awayTeam;
    }
    public void setAwayTeam(String awayTeam) {
        this.awayTeam = awayTeam;
    }
    public int getHomeTeamScore() {
        return homeTeamScore;
    }
    public void setHomeTeamScore(int homeTeamScore) {
        this.homeTeamScore = homeTeamScore;
    }
    public int getAwayTeamScore() {
        return awayTeamScore;
    }
    public void setAwayTeamScore(int awayTeamScore) {
        this.awayTeamScore = awayTeamScore;
    }
    public String homeTeam;
    public String awayTeam;
    public int homeTeamScore;
```

```
        public int awayTeamScore;


        public static Game createGameFromString(String s){

                Game g = new Game();
                int space = s.indexOf(" ");

                String homeTeam = s.substring(0, space);
                g.setHomeTeam(homeTeam);

                s = s.substring(space + 1);
                space = s.indexOf(" ");
                String awayTeam = s.substring(0, space);
                g.setAwayTeam(awayTeam);

                s = s.substring(space + 1);
                space = s.indexOf(" ");
                int homeTeamScore = Integer.valueOf(s.substring(0, space));
                g.setHomeTeamScore(homeTeamScore);

                s = s.substring(space + 1);
                space = s.indexOf(" ");
                int awayTeamScore = Integer.valueOf(s.substring(0,space));
                g.setAwayTeamScore(awayTeamScore);

                return g;
        }
}
```

Κλάση OutputNeuron.Java

```
import java.util.ArrayList;

public class OutputNeuron
{

        ArrayList<Double> inputValues = new ArrayList<Double>();
```

```
        ArrayList<Double> weightsValues = new ArrayList<Double>();
        ArrayList<Double> previewsWeihtsValues = new ArrayList<Double>();

        Double output;
        Double d= 0.0;
        Double previewsBiasWeight;
        Double biasWeight;
        Double sigma;


        OutputNeuron(int numberOfInp)
        {
               for(int i=0; i<numberOfInp; ++i)
               {
                      weightsValues.add(this.random(-1, 1));
               }
               this.sigma = this.random(0,1);
               output=0.0;
               biasWeight=this.random(0,1);
        }


        public double calculateOutput(ArrayList<Double> inValues)
        {
               inputValues=inValues;

               for(int i=0; i<inValues.size(); ++i)
               {
                      Double temp = inValues.get(i)*weightsValues.get(i);
                      output+=temp;
               }
               output+=biasWeight;
               output=sigMoidFunction(output)*1.5;


               return output;
        }


        void updateWeight(Double learningRate, ArrayList<Double>
inputArrayList,Double error)
        {
               previewsBiasWeight = biasWeight;

        }


        Double sigMoidFunction(Double value)
        {
               return (1.0/(1.0+Math.exp(-sigma*value)));
        }


        void computeD(Double inputForD)
        {
               d= output*inputForD;
        }
```

```
	public double random( double min, double max )
	{
	  double diff = max - min;
	  return min + Math.random( ) * diff;
	}
}
```

Κλάση CalculateScore.Java

```
import java.io.IOException;
import java.util.ArrayList;

public class CalculateScore
{


	public double calculate() throws IOException
	{
		System.out.println("The program trains.Please wait... ");
		ArrayList<String> errors = new ArrayList<String>();
		ReadTrainingFile trainOutput = new
ReadTrainingFile("training.txt", 11, 0);
		Double bias = random(0,1);

		Parameters pr = new Parameters();
		TestTraining testFile = new TestTraining(pr.testFile, 11);
		TestTraining trainFile = new TestTraining(pr.trainFile, 11);
		TestTraining centersFile = new TestTraining(pr.centersFile,
10);

		ArrayList<ArrayList<String>> allData = centersFile.dataset;
		Layer inputLayer = new Layer();
		Layer hidden = new Layer();


		ArrayList<String> tempOutputValues =
trainOutput.getColumn(10);

		ArrayList<Double> outputValues =
convertToDouble(tempOutputValues);

		for (int i = 0; i < Integer.valueOf(pr.noOfInputValues); i++)
		{
			inputLayer.inputNeurons.add(new InputNeuron());
		}



	  int hiddenLayerNeurons = Integer.valueOf(pr.noOfHiddenNeurons) ;
		}

		for (int i = 0; i < hiddenLayerNeurons; i++)
		{

			if(allData.size()!=0){
				ArrayList<String> temp = allData.get(i);
				ArrayList<Double> tmp=convertToDouble(temp);
				tmp.add(random(0,1));
```

```
                        hidden.neurons.add(new
Neuron(Integer.valueOf(pr.noOfInputValues),tmp));
                    }
                }

                for (int i = 0; i < Integer.valueOf(pr.iterations); i++)
                {
                        Double E = 0.0;
                        ArrayList<Double> allErrors = new ArrayList<Double>();

                        Double sumFinalValue = 0.0;
                        Double currentError = 1.0;
                        for (int j = 0; j < trainFile.getNumberOfLines(); j++)
                        {
                                sumFinalValue = 0.0;
                                currentError = 1.0;

                                ArrayList<Double> inputVector = new
ArrayList<Double>();
                                ArrayList<String> temp =
trainFile.getTheArray().get(j);

                                for (int z = 0; z < temp.size(); z++)
                                {
                                        InputNeuron tempInputNeuron =
inputLayer.inputNeurons
                                                        .get(z);

        tempInputNeuron.setInputs(Double.valueOf(temp.get(z)));
                                }

                                for (int z = 0; z <
inputLayer.inputNeurons.size(); z++)
                                {

        inputVector.add(inputLayer.inputNeurons.get(z).getValue());
                                }

                                for (int z = 0; z < hidden.neurons.size(); z++)
                                {
                                        Neuron neuronHidden =
hidden.neurons.get(z);
                                        neuronHidden.computeOutput(inputVector);
                                        sumFinalValue += neuronHidden.getOutput();
                                }


                                currentError = outputValues.get(j) -
sumFinalValue;
                                allErrors.add(currentError);

                                bias = bias +
Double.valueOf(pr.learningRateWeight) * currentError;

                                for (int z = 0; z < hidden.neurons.size(); z++)
                                {
                                        Neuron neuronHidden =
hidden.neurons.get(z);
```

```
neuronHidden.updateNetworkParameters(Double

.valueOf(pr.learningRateWeight), Double

.valueOf(pr.learningRateSigma), Double

.valueOf(pr.learningRateCenter), inputVector,
                                        currentError);

                        }
                }
                Double finalErr = 0.0;
                for (int j = 0; j < allErrors.size(); j++)
                {
                        Double temp = allErrors.get(j);
                        finalErr = finalErr + Math.pow(temp, 2);
                }
                E = finalErr / 2;
                errors.add(String.valueOf(i+"\t"+E+"\t"));

        }



        for (int j = 0; j < testFile.getNumberOfLines(); j ++)
        {
                ArrayList<Double> inputVector = new
ArrayList<Double>();

                ArrayList<String> temp = testFile.getTheArray().get(j);

                for (int z = 0; z < temp.size(); z++)
                {
                        InputNeuron tempInputNeuron =
inputLayer.inputNeurons
                                        .get(z);

        tempInputNeuron.setInputs(Double.valueOf(temp.get(z)));
                }

                for (int z = 0; z < inputLayer.inputNeurons.size(); z
++)
                {

        inputVector.add(inputLayer.inputNeurons.get(z).getValue());
                }


                ArrayList<Double> preoutput = new ArrayList<Double>();
                for (int z = 0; z < hidden.neurons.size(); z ++)
                {
                        Neuron neuronHidden = hidden.neurons.get(z);
                        preoutput.add(neuronHidden.getOutput());
                        neuronHidden.computeOutput(inputVector);

                }
```

```
                    OutputNeuron finalLayer = new OutputNeuron(
inputLayer.inputNeurons.size());

                    double result =
(int)Math.ceil(finalLayer.calculateOutput(preoutput)*
GenerateTest.absolutMaxScore);
                    return result;
             }


             return 0.0;

       }
       public static ArrayList<Double> convertToDouble(ArrayList<String>
input)
       {
             ArrayList<Double> reArray = new ArrayList<Double>();
             for (int i = 0; i < input.size(); ++i)
             {
                    reArray.add(Double.valueOf(input.get(i)));
             }

             return reArray;
       }


       public double random( double min, double max )
       {
         double diff = max - min;
         return min + Math.random( ) * diff;
       }

}
```

Κλάση MainClass.Java

```
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Scanner;

public class MainClass {
       public static String HOME_TEAM;
       public static String AWAY_TEAM;
       public static InputOutput io = new InputOutput();
       public static GenerateTest gt = new GenerateTest();
       public static CalculateScore c = new CalculateScore();
       public static double treshold = 200;


       public static void readInputFor2() {
             boolean valid = false;
             io = new InputOutput();
             ArrayList<String> teams = io
                          .readFileAsResource("resources/0405/TEAMS.txt");
```

```
            teams.add("OKLAHOMA");

            System.out
                        .println("Please enter the name of the home team,
with capital letters: ");

            do {

                  HOME_TEAM = new Scanner(System.in).nextLine();
                  if (teams.contains(HOME_TEAM)) {
                        valid = true;
                  } else {
                        System.err
                                    .println("No information about this
team. Please try again! ");
                        valid = false;
                  }

            } while (valid == false);
            System.out
                        .println("Please enter the name of the away team,
with capital letters: ");

            valid = false;
            do {

                  AWAY_TEAM = new Scanner(System.in).nextLine();
                  if (teams.contains(AWAY_TEAM)) {
                        valid = true;
                  } else {
                        System.err
                                    .println("No information about this
team. Please try again! ");
                        valid = false;
                  }

            } while (valid == false);
      }


      public static void main(String[] args) throws IOException {
            System.out.println("This program predicts the game results");
            System.out
                        .println("Please select what you want to do: 1
for predicting a specific game result; 2 for testing for the entire data
set.");
            int choice = 0;
            try {
                  choice = new Scanner(System.in).nextInt();
            } catch (Exception e) {
            }
            ;
            while ((choice != 1) && (choice != 2)) {
                  System.out
                              .println("You gave wrong input data.The
choice must be 1 or 2..");
                  System.out.print("Please make your choice:");
                  choice = new Scanner(System.in).nextInt();
            }
```

```
		if (choice == 1) {
			readInputFor2();

				System.out.println("generate for " + HOME_TEAM +
" "
					+ AWAY_TEAM);
				gt = new GenerateTest();
				gt.generateTestFile();
				c = new CalculateScore();
				double result = 0.0;
				try {
					result = c.calculate();
				} catch (IOException e) {
					e.printStackTrace();
				}
				System.out.println();
				System.out.println("Predicted result: " +
result);

				TestTraining testRead = null;
				try {
					testRead = new TestTraining("test.txt",
11);
				} catch (IOException e) {
					e.printStackTrace();
				}
				int finalScore =
Integer.valueOf(testRead.dataset.get(0).get(
					testRead.dataset.get(0).size() -
1));
				System.out.println("Correct result: " +
finalScore);
			}
		}

		if (choice == 2) {
			InputOutput io = new InputOutput();

			ArrayList<String> teams = io

	.readFileAsResource("resources/0405/TEAMS.txt");

			teams.add("OKLAHOMA");

			int correctRatio = 0;
			int totalGames = 0;
			try {

				FileWriter fw = new FileWriter("OVERALL.csv");
				PrintWriter pw = new PrintWriter(fw);
				pw.println("Home Team,Away
Team,Prediction,Actual,Correct");

				for (String host : teams) {
					for (String away : teams) {
						if ((((!host.equals("SEATTLE")) ||
(!away
							.equals("OKLAHOMA")))
```

```
                                                        &&
((!host.equals("OKLAHOMA")) || (!away

        .equals("SEATTLE")))) {

                                                    if (!(host.equals(away))) {
                                                        totalGames++;

                                                        HOME_TEAM = host;
                                                        AWAY_TEAM = away;

        System.out.println("generate for " + HOME_TEAM
                                                                + " " +
AWAY_TEAM);
                                                        gt = new
GenerateTest();
                                                        gt.generateTestFile();
                                                        c = new
CalculateScore();


                                                        double result = 0.0;
                                                        try {
                                                            result =
c.calculate();
                                                        } catch (IOException e)
{

        e.printStackTrace();
                                                        }

                                                        TestTraining testRead =
null;
                                                        try {
                                                            testRead = new
TestTraining("test.txt", 11);
                                                        } catch (IOException e)
{

        e.printStackTrace();
                                                        }
                                                        int finalScore =
Integer

        .valueOf(testRead.dataset.get(0)

        .get(testRead.dataset.get(0)

                    .size() - 1));

                                                        boolean correct =
false;

                                                        if (((result >=
treshold) && (finalScore >= treshold))
                                                                ||
((result < treshold) && (finalScore < treshold))) {
                                                            correct = true;
                                                            correctRatio++;
                                                        }
```

```
                                                        pw.println(HOME_TEAM +
","  + AWAY_TEAM + ","
                                                                + result +
","  + finalScore + ","
                                                                +
String.valueOf(correct));

                                                    }
                                                }
                                            }
                                        }

                                        pw.flush();

                                        pw.close();

                                        fw.close();

                                    }
                                    catch (IOException e) {
                                        System.err.println("Unable to write to file");
                                        System.exit(-1);
                                    }

                                    System.out.println("Finished.");
                                    System.out.println("The percentage of correct
predictions is: "
                                            + String.valueOf((double) correctRatio
                                                    / (double) totalGames));
                                    System.out
                                            .println("Detailed score results can be
found in OVERALL.csv");

                            }
                        }
                }
```