Ατομική Διπλωματική Εργασία

ΑΠΟΚΑΤΑΣΤΑΣΗ ΜΕΡΙΚΩΣ ΕΠΙΚΑΛΥΜΜΕΝΩΝ ΕΙΚΟΝΩΝ ΠΡΟΣΩΠΟΥ ΜΕ ΔΙΚΤΥΑ HOPFIELD ΚΑΙ ΜΗΧΑΝΕΣ BOLTZMANN

Αποστόλου Λένα

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Ιούνιος 2008

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Αποκατάσταση μερικώς επικαλυμμένων εικόνων προσώπου με δίκτυα Hopfield και μηχανές Boltzmann

Αποστόλου Λένα

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

Ιούνιος 2008

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Δρ. Χρίστο Χριστοδούλου, που μου έδωσε την ευκαιρία να εκπονήσω τη συγκεκριμένη διπλωματική. Με τη βοήθεια και τη σωστή καθοδήγηση που μου παρείχε όλο αυτό το διάστημα, καθώς και με την υπομονή που έχει υποδείξει, κατάφερα να φέρω εις πέρας μεγάλο κομμάτι της εργασίας.

Θα ήθελα, επίσης, να ευχαριστήσω ιδιαιτέρως τη Δρ. Chrisina Draganova, για την εξαιρετικά πολύτιμη βοήθεια που μου έχει προσφέρει.

Περίληψη

Η αποκατάσταση μερικώς επικαλυμμένων εικόνων προσώπου με νευρωνικά δίκτυα είναι ένα αρκετό ενδιαφέρον θέμα στο χώρο της πληροφορικής. Έχουν γίνει αρκετές μελέτες που επιλύουν το συγκεκριμένο πρόβλημα με διάφορα είδη νευρωνικών δικτύων και τα αποτελέσματα τους είναι ικανοποιητικά.

Στη διπλωματική αυτή γίνεται προσπάθεια για επίλυση του προβλήματος αποκατάστασης μερικώς επικαλυμμένων εικόνων προσώπου με νευρωνικά δίκτυα Hopfield και μηχανές Boltzmann. Τα δύο δίκτυα εκπαιδεύονται με τις ίδιες εικόνες προσώπου και ακολούθως παρουσιάζονται σε αυτά διάφορες θορυβώδεις ή ελλιπείς εικόνες προσώπου. Στόχος είναι να ανακτήσουν την κοντινότερη ενεργειακά εικόνα προσώπου.

Σύμφωνα με τα αποτελέσματα της διπλωματικής αυτής αλλά και με βάση τη θεωρία, το δίκτυο Hopfield είναι σχετικά αποδοτικό όσον αφορά την επίλυση του πιο πάνω προβλήματος, διότι αντιμετωπίζει σημαντικά προβλήματα. Ο μέσος όρος του σφάλματος, όπως υπολογίστηκε κατά τον έλεγχο του δικτύου Hopfield, κυμαίνεται γύρω στο 1.5. Οι μηχανές Boltzmann, σύμφωνα με τη θεωρία, πρέπει να έχουν πιο αποδοτικά αποτελέσματα, εφόσον αποτελούν επέκταση του δικτύου Hopfield και έχουν ακριβώς δημιουργηθεί για να αντιμετωπίσουν τα προβλήματα που αντιμετωπίζουν τα δίκτυα Hopfield. Η καθυστέρηση όμως ολοκλήρωσης του δικτύου Hopfield και το γεγονός ότι οι μηχανές Boltzmann είναι πάρα πολύ αργό δίκτυο, δεν μας άφησε χρόνο να αποδείξουμε τη θεωρία. Το δίκτυο Boltzmann έχει ναι μεν ολοκληρωθεί, χωρίς όμως τα αναμενόμενα αποτελέσματα. Ο λίγος χρόνος που είχε απομένει δεν ήταν αρκετός για να εκπαιδευτεί σωστά το δίκτυο και να έχει πιο ορθά αποτελέσματα.

Επίσης, η εργασία αυτή βασίζεται σε μια μελέτη που έγινε από τους Draganova et al. [4]. Η μελέτη αυτή επιλύει το ίδιο θέμα με διαφορετικές τεχνικές και γι'αυτό το λόγο συγκρίνουμε τα αποτελέσματα των δύο μελετών.

Περιεχόμενα

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός και σημασία του προβλήματος

Η αναγνώριση και αποκατάσταση μερικώς επικαλυμμένων εικόνων προσώπου είναι ένα από τα θέματα που προσελκύουν μεγάλο επιστημονικό ενδιαφέρον στο χώρο της πληροφορικής. Πολλοί μελετητές έχουν ασχοληθεί με το συγκεκριμένο θέμα τα τελευταία είκοσι χρόνια και πολλές μέθοδοι έχουν προταθεί. Μέχρι σήμερα το ενδιαφέρον πολλών ερευνητών παραμένει αναλλοίωτο για το συγκεκριμένο ζήτημα, εφόσον η χρήση του είναι ευρεία και σημαντική σε πολλούς τομείς, όπως στην αναγνώριση προτύπων, στην ψυχολογία, στην επεξεργασία εικόνας και στα γραφικά υπολογιστών [9].

Πολλές εφαρμογές έχουν ως απαίτηση αποδοτική αποκατάσταση εικόνων προσώπου που ίσως να επικαλύπτονται από διάφορα αντικείμενα, όπως γυαλιά, γένια ή καπέλα. Για παράδειγμα, εφαρμογές για αλληλεπίδραση ανθρώπου-ρομπότ και ανθρώπου-υπολογιστή, όπου χρειάζεται η αναγνώριση της ανθρώπινης κίνησης. Ή εφαρμογές όπως ασφάλεια δεδομένων και πληροφοριών. Για μια ασφαλή σύνδεση σε μια μηχανή ή στο

διαδίκτυο ή για ασφάλεια εφαρμογών και βάσεων δεδομένων, η αναγνώριση προσώπου θα είναι αρκετά αποδοτική και δεν θα χρειάζεται καν η συνεργασία ή ακόμα και η ενημέρωση του χρήστη. Τέλος, η αναγνώριση εικόνων προσώπου είναι πολύ σημαντική στην επιβολή του νόμου, σε περιπτώσεις δηλαδή κλοπών από διάφορα μαγαζιά και γενικά στην έρευνα και παρακολούθηση υπόπτων [9].

Η αναγνώριση προσώπων είναι μια λειτουργία που επιτυγχάνεται πολύ εύκολα από ένα άνθρωπο και πολύ δύσκολα από μια μηχανή, συγκεκριμένα από ένα αλγόριθμο επεξεργασίας εικόνων προσώπου. Όταν ένας άνθρωπος έρθει σε επαφή με ένα αντικείμενο που είναι επικαλυμμένο από άλλα αντικείμενα, τότε μπορεί εύκολα να διακρίνει το σχήμα του αρχικού αντικειμένου χωρίς την επικάλυψη. Αν το αντικείμενο είναι οικείο στον άνθρωπο, τότε μπορεί να διακρίνει το σχήμα του ακόμα και αν το επικαλυμμένο μέρος του αντικειμένου είναι πολύπλοκο. Στην περίπτωση που το αντικείμενο δεν είναι οικείο, ο άνθρωπος μπορεί επίσης να φανταστεί και να διακρίνει το αρχικό του σχήμα με βάση τη γεωμετρία των μη επικαλυμμένων μερών του [6].

Η ανθρώπινη αυτή διαδικασία αναγνώρισης προσώπων μοντελοποιείται από διάφορα νευρωνικά δίκτυα. Εκπαιδεύεται το νευρωνικό δίκτυο με διάφορες εικόνες προσώπου και μετά όταν παρουσιάσουμε στο δίκτυο μια θορυβώδης ή ελλιπής εικόνα προσώπου, τότε αυτό θα ανακτήσει την κοντινότερη ενεργειακά εικόνα, όπως ακριβώς συμβαίνει και με τον ανθρώπινο εγκέφαλο. Όταν ο άνθρωπος έρθει σε επαφή με μια θορυβώδης ή ελλιπής εικόνα προσώπου, τότε θα ανακτήσει στη μνήμη του είτε την ίδια την εικόνα είτε την πιο κοντινή ενεργειακά.

Έχοντας υπόψη μας τη δυνατότητα των νευρωνικών δικτύων να μοντελοποιούν την ανθρώπινη λειτουργία αναγνώρισης αντικειμένων, θα χρησιμοποιήσουμε δύο συγκεκριμένα μοντέλα νευρωνικών δικτύων, τα δίκτυα Hopfield [8] και τις μηχανές Boltzmann [1] για να επιλύσουμε το πρόβλημα αποκατάστασης μερικώς επικαλυμμένων εικόνων προσώπου. Ο κύριος λόγος που θα χρησιμοποιήσουμε τα δύο αυτά δίκτυα είναι η ιδιότητα τους να συσχετίζουν πρότυπα, το πρόβλημα ακριβώς που πρέπει να λυθεί στο συγκεκριμένο θέμα αποκατάστασης επικαλυμμένων εικόνων. Τα δύο αυτά δίκτυα θα

εκπαιδευτούν με εικόνες προσώπου από διάφορα άτομα και ακολούθως θα παρουσιάζονται σε αυτά θορυβώδεις και ελλιπείς εικόνες. Σκοπός είναι να ανακτηθούν οι εικόνες που είναι ενεργειακά πιο κοντά στις εικόνες που παρουσιάζονται στο δίκτυο.

1.2 Περιγραφή προηγούμενης εργασίας

Όπως έχει ήδη προαναφερθεί, το θέμα της αποκατάστασης επικαλυμμένων εικόνων και γενικά επικαλυμμένων αντικειμένων απασχολεί ερευνητές από διάφορους τομείς. Πολλές σημαντικές προσπάθειες έχουν γίνει στο παρελθόν με στόχο την επίλυση του συγκεκριμένου προβλήματος με διάφορες τεχνικές και μεθόδους. Παρακάτω θα περιγράψουμε εν συντομία μερικές από αυτές.

Η παρούσα διπλωματική βασίζεται σε μια μελέτη που έχει ήδη γίνει από τους Draganova et al. [4] και αφορά το ίδιο ακριβώς θέμα αποκατάστασης μερικώς επικαλυμμένων εικόνων προσώπου. Στη συγκεκριμένη μελέτη έχουν χρησιμοποιηθεί άλλα μοντέλα νευρωνικών δικτύων και έχει αποδειχθεί μέσω των αποτελεσμάτων ότι είναι δυνατό να αποκαταστήσουμε με αρκετή ακρίβεια εικόνες προσώπου οι οποίες έχουν ένα μεγάλο μέρος τους επικαλυμμένο ή είναι αρκετά θορυβώδεις.

Οι μέθοδοι που χρησιμοποιήθηκαν για την επίλυση του προβλήματος είναι τα νευρωνικά δίκτυα Hopfield, τα νευρωνικά δίκτυα πολλαπλών στρωμάτων Perceptron (MLP), μια καινούρια τεχνική που συνδυάζει δίκτυα Hopfield και MLP και μια μέθοδος βασισμένη στην συσχετιζόμενη αναζήτηση (associative search). Σύμφωνα με τα αποτελέσματα της μελέτης, ο συνδυασμός των δικτύων Hopfield και πολλαπλών στρωμάτων Perceptron δίνει τα καλύτερα αποτελέσματα και το ελάχιστο σφάλμα, εν αντιθέσει με τα δίκτυα Hopfield που έχουν τα πιο ανακριβή αποτελέσματα και το μεγαλύτερο σφάλμα.

Μια άλλη μελέτη έγινε από τους Kim Blackwell et al. [3] και αφορά την αναγνώριση εικόνων προσώπου με θόρυβο και τη σύγκριση της αποδοτικότητας του Νευρωνικού Δικτύου Dystal με τους ανθρώπους παρατηρητές. Το νευρωνικό δίκτυο Dystal είναι ένας αλγόριθμος συσχετιζόμενης μάθησης (associative learning algorithm), που προέρχεται

από τη βιοφυσική και την ηλεκτροφυσιολογία της συσχετιζόμενης μάθησης. Οι εικόνες προσώπου αποθηκεύονται στο δίκτυο σαν μέσος όρος παρόμοιων εικόνων που είναι ήδη αποθηκευμένες. Η ομοιότητα μεταξύ της εικόνας εισόδου και των αποθηκευμένων εικόνων υπολογίζεται με την Pearson's R συσχέτιση, η οποία εφαρμόζεται σε ολόκληρη την εικόνα, σε κάθε ένα εικονοστοιχείο ξεχωριστά (global processing). Τα δεδομένα που χρησιμοποιήθηκαν στον έλεγχο του δικτύου διέφεραν από τα δεδομένα εκπαίδευσης ως προς τις εκφράσεις προσώπου, την κλίση και περιστροφή του κεφαλιού και την παρουσία θορύβου στις εικόνες. Το δίκτυο Dystal και οι άνθρωποι καλούνται να αναγνωρίσουν αυτές τις εικόνες που διαφέρουν από τις εικόνες εκπαίδευσης. Η γενική ιδέα της συγκεκριμένης μελέτης περιστρέφεται γύρω από το γεγονός ότι οι άνθρωποι αναγνωρίζουν εικόνες προσώπου μέσω της εξαγωγής χαρακτηριστικών. Αυτό σημαίνει ότι η απόδοση αναγνώρισης μειώνεται όταν υπάρχει θόρυβος στην εικόνα. Αντιθέτως, το δίκτυο Dystal εκτελεί ολική επεξεργασία (global processing) και έτσι αν υπάρχει θόρυβος στην εικόνα, η απόδοση αναγνώρισης είναι μεγαλύτερη εν συγκρίσει με αυτή των ανθρώπων. Τα αποτελέσματα της μελέτης αυτής απέδειξαν ότι η απόδοση του Νευρωνικού Δικτύου Dystal είναι συγκρίσιμη με αυτή του ανθρώπου. Ο άνθρωπος ήταν πιο αποδοτικός με εικόνες χωρίς θόρυβο, ενώ το δίκτυο Dystal ήταν ικανό να αναγνωρίσει εικόνες προσώπου με θόρυβο καλύτερα από τον άνθρωπο.

Ο Kunihiko Fukushima [6] ασχολήθηκε με την αποκατάσταση μερικώς επικαλυμμένων προτύπων και προτείνει ένα μοντέλο νευρωνικού δικτύου, το οποίο έχει τη δυνατότητα να αποκαθιστά τα μέρη που δεν φαίνονται σε ένα μερικώς επικαλυμμένο πρότυπο. Πρόκειται για ένα ιεραρχικό δίκτυο πολλαπλών στρωμάτων με εμπρόσθια (forward) και οπίσθια (backward) μονοπάτια σημάτων (signal paths). Τα επικαλυμμένα μέρη ενός προτύπου αποκαθιστούνται κυρίως μέσω των σημάτων από τις ψηλότερες καταστάσεις του δικτύου, ενώ τα μη επικαλυμμένα μέρη του προτύπου παράγονται ξανά μέσω των σημάτων από τις χαμηλότερες καταστάσεις του δικτύου. Σύμφωνα με τα αποτελέσματα, το προτεινόμενο μοντέλο έχει τη δυνατότητα να αναγνωρίσει και να αποκαταστήσει σωστά ένα επικαλυμμένο πρότυπο, αν το δίκτυο έχει εκπαιδευτεί με το συγκεκριμένο πρότυπο. Αν το δίκτυο δεν έχει εκπαιδευτεί με το πρότυπο ελέγχου, τότε προσπαθεί να το συμπληρώσει χρησιμοποιώντας παρεμβολή (interpolation) και extrapolation των

ορατών ακμών. Σε μια προσομοίωση όμως που αναφέρεται στο άρθρο, τα πρότυπα είχαν χωριστεί σε κομμάτια ανάλογα με τη φωτεινότητα τους και αυτό δεν έχει πάντα ικανοποιητικά αποτελέσματα, όπως τονίζει ο συγγραφέας.

Κεφάλαιο 2

Δεδομένα εισόδου

2.1 Σύντομη περιγραφή δεδομένων εισόδου

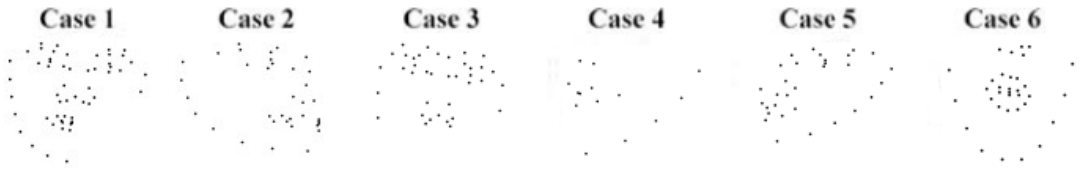
Τα δεδομένα εισόδου που χρησιμοποίησα στη διπλωματική για την εκπαίδευση και τον έλεγχο των νευρωνικών δικτύων είναι εικόνες προσώπου που περιέχονται στη βάση δεδομένων FG-NET Aging Database [10]. Τα δεδομένα αυτά έχουν χρησιμοποιηθεί και στη μελέτη των Draganova et al. [4]. Στο σχήμα 2.1 φαίνονται μερικά τυπικά παραδείγματα εικόνων προσώπου. Στη βάση δεδομένων υπάρχουν συνολικά 1002 εικόνες προσώπου από 82 διαφορετικά άτομα. Για την εκπαίδευση των νευρωνικών δικτύων χρησιμοποιήθηκαν 102 εικόνες προσώπου, που αντιστοιχούν σε περίπου 8 άτομα. Ο λόγος που χρησιμοποιήθηκαν μόνο 102 πρότυπα είναι, όπως θα δούμε και πιο κάτω στο υποκεφάλαιο 3.1, ο περιορισμός που έχουν τα δίκτυα Hopfield όσον αφορά τη χωρητικότητα τους. Για τον έλεγχο των νευρωνικών δικτύων χρησιμοποιήθηκαν οι υπόλοιπες 900 εικόνες προσώπου.

Σχήμα 2.1: Παραδείγματα εικόνων προσώπου από τη βάση δεδομένων FG-NET Aging Database [10]

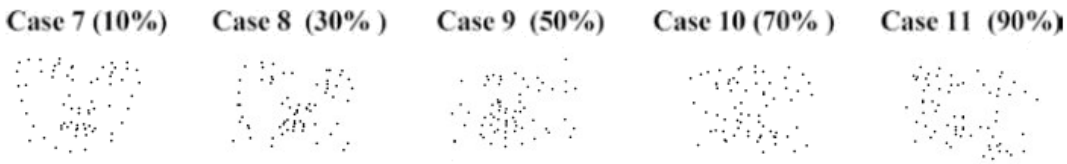
Στην υλοποίηση, κάθε εικόνα προσώπου παριστάνεται από συντεταγμένες 68 σημείων, τα οποία αντιπροσωπεύουν το γενικό περίγραμμα του προσώπου και το περίγραμμα των επιμέρους χαρακτηριστικών του προσώπου. Η θέση των σημείων αυτών στο πρόσωπο φαίνεται στο σχήμα 2.2. Η αναπαράσταση κάθε εικόνας με σημεία βρίσκεται επίσης στη βάση δεδομένων FG-NET Aging Database [10].

Σχήμα 2.2: Κάθε εικόνα προσώπου καθορίζεται από 68 σημεία

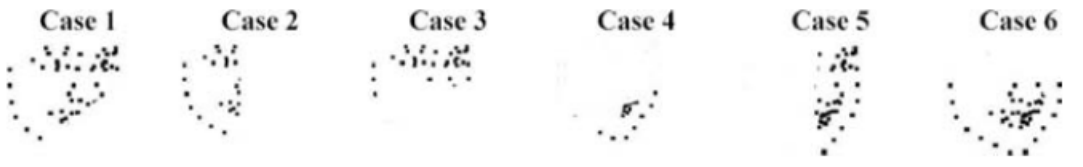
Τα δεδομένα ελέγχου χωρίστηκαν σε δύο γενικές κατηγορίες, όπως και στη μελέτη [4], στις εικόνες με θόρυβο και στις επικαλυμμένες εικόνες. Στην πρώτη κατηγορία υπάρχουν περιπτώσεις 10%, 30%, 50%, 70%, 90% θορύβου και στη δεύτερη κατηγορία υπάρχουν διάφορες περιπτώσεις στις οποίες επικαλύπτεται διαφορετικό μέρος του προσώπου κάθε φορά. Στα σχήματα 2.3 και 2.4 παρουσιάζονται οι δύο κατηγορίες και οι διάφορες περιπτώσεις της παρούσας διπλωματικής. Στη μελέτη [4], τα σημεία που επικαλύπτονται σε μια εικόνα προσώπου ή τα σημεία με θόρυβο είναι επιλεγμένα με διαφορετικό τρόπο, ο οποίος φαίνεται στα σχήματα 2.5 και 2.6.



Σχήμα 2.3: Περιπτώσεις 1-6 παρούσας διπλωματικής: Επικάλυψη διαφορετικών μερών μιας εικόνας προσώπου



Σχήμα 2.4: Περιπτώσεις 7-11 παρούσας διπλωματικής: Θορυβώδεις εικόνες προσώπου. Ο θόρυβος σε κάθε περίπτωση δημιουργείται με την αντικατάσταση ενός αυξανόμενου αριθμού σημείων με τυχαίους αριθμούς



Σχήμα 2.5: Περιπτώσεις 1-6 μελέτης [4]: Επικάλυψη διαφορετικών μερών μιας εικόνας προσώπου

Σχήμα 2.6: Περιπτώσεις 7-11 μελέτης [4]: Θορυβώδεις εικόνες προσώπου. Ο θόρυβος σε κάθε περίπτωση δημιουργείται με την αντικατάσταση ενός αυξανόμενου αριθμού σημείων με τυχαίους αριθμούς

Κεφάλαιο 3

Γνωσιολογικό Υπόβαθρο

3.1 Δίκτυα Hopfield

Νευρωνικά Δίκτυα που εφευρέθηκαν από τον John Hopfield [8]. Το δίκτυο Hopfield είναι ένα δυαδικό τεχνητό δίκτυο, το οποίο έχει τη δυνατότητα να αποθηκεύει πρότυπα με έναν τρόπο παρόμοιο με αυτό του εγκεφάλου - αν στο δίκτυο παρουσιαστεί θορυβώδης ή ελλιπής πληροφορία, τότε ανακτάται ολόκληρο το αρχικό πρότυπο αν αυτό είναι ήδη αποθηκευμένο, διαφορετικά ανακτάται το πιο ενεργειακά κοντινότερο αποθηκευμένο πρότυπο. Με ανάλογο τρόπο λειτουργεί και η ανθρώπινη μνήμη: δεδομένης μιας δυσδιάκριτης εικόνας, συσχετίζουμε το περιεχόμενο της με κάποια πρότυπα που έχουμε ήδη στη μνήμη μας και ανακτούμε το πιο «κοντινό» πρότυπο.

Το δίκτυο Hopfield μπορεί, επίσης, να θεωρηθεί και σαν συσχετιζόμενη μνήμη, εφόσον το δίκτυο θα συσχετίσει το ελλιπής ή θορυβώδης πρότυπο, που του δίνεται σαν είσοδος, με τα ήδη αποθηκευμένα πρότυπα και θα ανακτήσει αυτό που το προσεγγίζει καλύτερα. Το σημαντικό αυτό χαρακτηριστικό του δικτύου οφείλεται στην ικανότητα του να φιλτράρει τα θορυβώδη ή ελλιπή πρότυπα εισόδου.

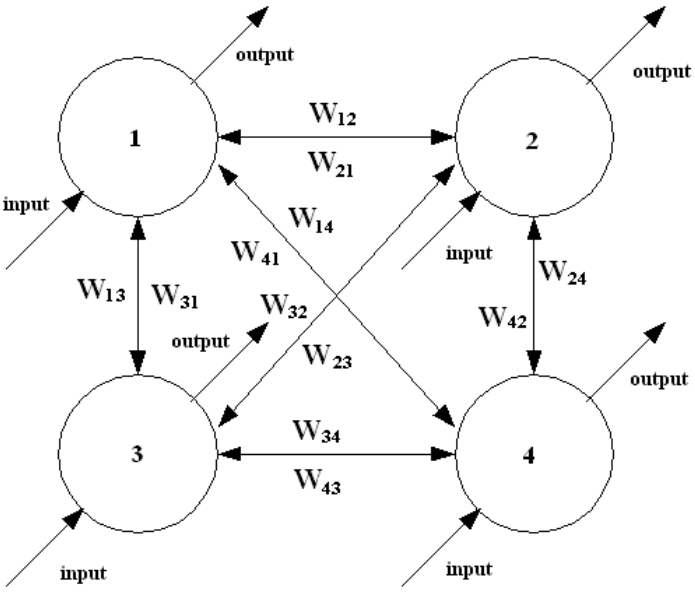
Ένα δίκτυο Hopfield είναι ένα πλήρως συνδεδεμένο δίκτυο, όπως φαίνεται και στο σχήμα 3.1 και αποτελείται από ένα αριθμό νευρώνων. Ο κάθε νευρώνας είναι συνδεδεμένος με όλους τους άλλους νευρώνες και για κάθε σύνδεση αντιστοιχεί ένα βάρος w_{ij}. Η ανάθεση των βαρών w_{ij} μεταξύ των νευρώνων i και j, υποδηλώνει την ισχύ και τη δύναμη της σύνδεσης από το νευρώνα i στον νευρώνα j. Σύμφωνα με τη δομή του δικτύου, τα βάρη πρέπει να είναι συμμετρικά, δηλαδή το βάρος w_{ij} που συνδέει τον νευρώνα i με τον νευρώνα j να είναι ίσο με το βάρος w_{ji} που συνδέει τον νευρώνα j με τον νευρώνα i. Επίσης, το βάρος $w_{ii} = 0$, διότι ο κάθε νευρώνας δεν έχει σύνδεση που να καταλήγει στον εαυτό του. Τα βάρη υπολογίζονται με βάση τον τύπο που ακολουθεί:

$$w_{ij} = \sum_{r=1}^{M} p_i^r \, p_j^r \quad \text{αν } i \neq j$$

$$w_{ij} = 0 \quad \text{αν } i = j$$

όπου:

- $(i,j) \in (0,N-1)$
- p_i^r $(r = 1,2,\ldots,M)$ = έξοδος του νευρώνα i
- M = αριθμός αποθηκευμένων προτύπων
- N = αριθμός νευρώνων που αποτελούν το δίκτυο



Σχήμα 3.1: Δομή δικτύου Hopfield που αποτελείται από τέσσερις νευρώνες

Ένας νευρώνας μπορεί να βρίσκεται σε δύο πιθανές καταστάσεις, είτε είναι ενεργός είτε μη ενεργός, δηλαδή είτε η έξοδος του είναι +1 είτε -1. Η αλλαγή στην κατάσταση ενός νευρώνα είναι ανεξάρτητη από την προηγούμενη κατάσταση και γίνεται με βάση τη συνάρτηση κατωφλίου Heaviside. Η έξοδος κάθε νευρώνα στέλνεται σαν είσοδος σε όλα τα υπόλοιπα νευρώνια και τότε αυτά με τη σειρά τους πρέπει να ορίσουν εκ νέου την κατάσταση τους. Σε κάθε νευρώνα αντιστοιχεί ένα κατώφλι και με βάση αυτού και της εισόδου του νευρώνα αποφασίζεται αν η έξοδος θα είναι +1/-1. Συνοπτικά, η ενεργοποίηση και η απενεργοποίηση ενός νευρώνα επιτυγχάνεται αντίστοιχα με τον πιο κάτω τύπο:

$$\mathbf{O_i\,(t+1) = sgn\ (\sum_{j=1}^{N} W_{ij} O_j\,(t)\ -\ \Theta_i\ )}$$

όπου:

- **sgn**(x) = 1 αν $x \geq 0$ ή **sgn**(x) = -1 αν $x < 0$
- $O_i(t+1)$ = έξοδος του νευρώνα i στο χρόνο t+1
- $O_j(t)$ = έξοδος του νευρώνα j στο χρόνο t
- Θ_i = κατώφλι του νευρώνα i
- N = αριθμός νευρώνων που αποτελούν το δίκτυο
- W_{ij} = το βάρος μεταξύ των νευρώνων i και j

Όταν παρουσιάσουμε στο δίκτυο μας ένα καινούριο πρότυπο, τότε αυτό ταλαντεύεται, αλλάζει δηλαδή συνεχώς καταστάσεις, μέχρι να καταλήξει στο πιο κοντινό ενεργειακά πρότυπο και να το ανακτήσει. Όταν ένα δίκτυο ταλαντεύεται, σημαίνει ότι οι νευρώνες που αποτελούν το δίκτυο αλλάξουν συνεχώς καταστάσεις από +1 σε -1 ή αντίθετα από -1 σε +1. Όταν όλοι οι νευρώνες σταματήσουν να αλλάζουν καταστάσεις και οι επόμενες καταστάσεις τους είναι ίδιες με τις προηγούμενες, δηλαδή από την κατάσταση +1 παραμείνουν στην κατάσταση +1 και από την κατάσταση -1 παραμείνουν στην κατάσταση -1, τότε σημαίνει το δίκτυο μας έχει καταλήξει σε ένα πρότυπο. Κατά τη διάρκεια της ταλάντωσης, οι νευρώνες αλλάζουν κατάσταση σύμφωνα με τον πιο πάνω τύπο με ασυγχρόνιστο τρόπο. Δηλαδή, σε κάθε επανάληψη επιλέγεται τυχαία ο νευρώνας ο οποίος θα αλλάξει κατάσταση. Μετά από αρκετές επαναλήψεις – ταλαντώσεις και

ανεξάρτητα από την αρχική κατάσταση του, το δίκτυο καταλήγει στο ενεργειακά μικρότερο πρότυπο. Η έξοδος του δικτύου, το πρότυπο δηλαδή που ανακτήθηκε, είναι το πρότυπο που έχει το μικρότερο Hamming Distance από το πρότυπο εισόδου, το πρότυπο δηλαδή που έχει τα λιγότερα διαφορετικά ψηφία από το πρότυπο εισόδου.

Τα δίκτυα Hopfield παρουσιάζουν δύο κύρια προβλήματα. Το πρώτο πρόβλημα αφορά τη χωρητικότητα του δικτύου. Ο μέγιστος αριθμός προτύπων που μπορούν να αποθηκευτούν και να ανακτηθούν με πλήρης ακρίβεια είναι περίπου το 0.15*N, όπου N είναι ο αριθμός των νευρώνων που αποτελούν το δίκτυο. Για παράδειγμα, ένα δίκτυο με 100 κόμβους, θα μπορεί να αποθηκεύσει μόνο μέχρι 15 πρότυπα ούτως ώστε να έχει ικανοποιητικά αποτελέσματα.

Το σημαντικότερο όμως πρόβλημα των δικτύων Hopfield, είναι τα τοπικά ελάχιστα. Το πρόβλημα αυτό προκύπτει όταν το δίκτυο σταματήσει και παραμείνει σε μια κατάσταση που δεν είναι η μικρότερη ενεργειακά. Τότε, ανακτάται το πρότυπο που είναι αποθηκευμένο σε αυτό το τοπικό ελάχιστο, που όμως δεν είναι απαραίτητα και το πιο σωστό. Αυτό συμβαίνει διότι το δίκτυο ξεκινά να ταλαντεύεται κοντά σε κάποιο τοπικό ελάχιστο και η αναμενόμενη συμπεριφορά του είναι να καταλήξει σε αυτό το ελάχιστο και όχι να βρει το ολικό ελάχιστο στο οποίο βρίσκεται το ενεργειακά μικρότερο πρότυπο. Η ύπαρξη των τοπικών ελαχίστων έχει καλά αποτελέσματα όσον αφορά την συσχέτιση προτύπων, όχι όμως όταν χρειαζόμαστε βελτιστοποίηση.

3.2 Μηχανές Boltzmann

Οι μηχανές Boltzmann είναι νευρωνικά δίκτυα που βασίζονται σε στοχαστικό τρόπο μάθησης και εφευρέθηκαν από τους Terrence J. Sejnowski και Geoffrey E. Hinton [1]. Αποτελούν επέκταση των δικτύων Hopfield και η δημιουργία τους στηρίζεται στην προσπάθεια αντιμετώπισης των προβλημάτων που παρουσιάζουν τα δίκτυα Hopfield. Τα κοινά χαρακτηριστικά των δύο αυτών δικτύων είναι τα εξής:

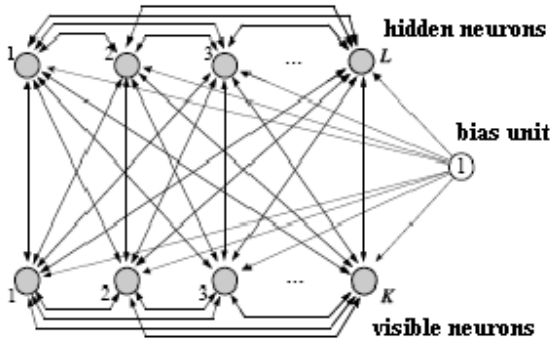
1. Οι καταστάσεις των νευρώνων παίρνουν τιμές +1/-1.
2. Όλες οι συνάψεις μεταξύ των νευρώνων είναι συμμετρικές.

3. Οι νευρώνες που πρόκειται να αλλάξουν κατάσταση κατά τη διάρκεια ταλάντωσης του δικτύου επιλέγονται τυχαία.
4. οι νευρώνες δεν έχουν αυτοεπανατροφοδότηση

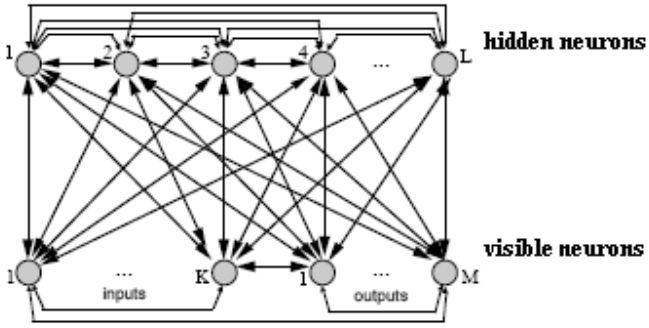
Εντούτοις, διαφέρουν σε τρία πιο σημαντικά θέματα:

1. Στις μηχανές Boltzmann έχουν προστεθεί κρυφά νευρώνια έτσι ώστε να αποφευχθεί το πρόβλημα χωρητικότητας που υπάρχει στα δίκτυα Hopfield.
2. Οι νευρώνες στις μηχανές Boltzmann χρησιμοποιούν στοχαστική συνάρτηση κατωφλίου βασισμένη σε πιθανότητες [7], ενώ στα δίκτυα Hopfield οι νευρώνες βασίζονται στο McCulloch-Pitts μοντέλο.
3. Στο δίκτυο Hopfield έχουμε μη επιβλεπόμενη μάθηση, ενώ οι μηχανές Boltzmann μπορούν να εκπαιδευτούν και με επιβλεπόμενη μάθηση.

Υπάρχουν δύο βασικές αρχιτεκτονικές των μηχανών Boltzmann, το δίκτυο ολοκλήρωσης Boltzmann και το δίκτυο εισόδου – εξόδου Boltzmann. Το δίκτυο ολοκλήρωσης Boltzmann έχει τη δυνατότητα να μαθαίνει μια σειρά από πρότυπα εισόδου και ακολούθως να συμπληρώνει τα υπολειπόμενα μέρη των προτύπων, όταν στο δίκτυο μας παρουσιαστούν θορυβώδης ή ελλιπής πρότυπα. Σε αυτή την αρχιτεκτονική δικτύου υπάρχουν δύο είδη νευρώνων, οι ορατοί και οι κρυφοί νευρώνες. Αντίθετα, το δίκτυο εισόδου – εξόδου Boltzmann εκτελεί συσχέτιση με επιβλεπόμενη μάθηση. Η ιδιότητα του αυτή επιτυγχάνεται μέσω της διάταξης του. Στο δίκτυο εισόδου – εξόδου Boltzmann η διαφορά είναι ότι οι ορατοί νευρώνες κατηγοριοποιούνται σε νευρώνες εισόδου και νευρώνες εξόδου. Οι νευρώνες εισόδου λαμβάνουν τις απαραίτητες πληροφορίες από το εξωτερικό περιβάλλον και οι νευρώνες εξόδου αναφέρουν το αποτέλεσμα των υπολογισμών του δικτύου πάλι στο εξωτερικό περιβάλλον. Στα σχήματα 3.2 και 3.3 φαίνονται οι δομές των δύο αυτών αρχιτεκτονικών. Στη διπλωματική αυτή θα χρησιμοποιήσουμε δίκτυο εισόδου – εξόδου Boltzmann, λόγω της συσχέτισης προτύπων που έχει τη δυνατότητα να μας παρέχει.



Σχήμα 3.2: Δομή δικτύου ολοκλήρωσης Boltzmann



Σχήμα 3.3: Δομή δικτύου εισόδου – εξόδου Boltzmann

Η γενική ιδέα για τις μηχανές Boltzmann βασίζεται στη μέθοδο της προσομοιωμένης ψύξης. Η στοχαστική αυτή τεχνική βοηθά στην αντιμετώπιση του σοβαρότερου προβλήματος των δικτύων Hopfield, στην αποφυγή δηλαδή των τοπικών ελαχίστων και στην εύρεση των ολικών ελαχίστων. Σύμφωνα με τη μέθοδο της προσομοιωμένης ψύξης, αφήνουμε το δίκτυο μας να ταλαντωθεί σε ψηλή θερμοκρασία και ακολούθως, μειώνουμε σταδιακά την θερμοκρασία αυτή μέχρι το δίκτυο να φτάσει σε θερμική ισορροπία [5]. Κατά τη θερμική ισορροπία το δίκτυο συνεχίζει να ταλαντεύεται και τα νευρώνια να αλλάζουν καταστάσεις, με σταθερές όμως πιθανότητες. Η χρησιμότητα της προσθήκης θερμοκρασίας στο δίκτυο είναι για να το αναγκάσει να ταλαντεύεται περισσότερο και έτσι να ξεφεύγει από τα τοπικά ελάχιστα. Για να σταματήσει να ταλαντεύεται και να καταλήξει σε κάποιο πρότυπο στο τέλος, πρέπει να εφαρμόσουμε την προσομοίωση ψύξης. Με την εφαρμογή της μεθόδου αυτής, όσο το δίκτυο πλησιάζει

στη θερμική ισορροπία, οι χαμηλές ενεργειακές καταστάσεις είναι πιο πιθανές [2] (η απόδειξη για το ότι οι χαμηλότερες ενεργειακά καταστάσεις είναι πιθανότερες όσο το δίκτυο πλησιάζει στη θερμική ισορροπία βρίσκεται στο παράρτημα Α). Άρα οι πιθανότητες να καταλήξει το δίκτυο μας στο ενεργειακά κοντινότερο πρότυπο είναι μεγαλύτερες.

Η εκπαίδευση ενός δικτύου Boltzmann χωρίζεται σε δύο φάσεις. Στη πρώτη φάση παγιώνουμε τα νευρώνια εισόδου και εξόδου με το αντίστοιχο πρότυπο εισόδου και την αντίστοιχη επιθυμητή έξοδο, ενώ τα κρυφά νευρώνια αρχικοποιούνται με τυχαίες τιμές. Ακολούθως, εφαρμόζουμε την μέθοδο της προσομοιωμένης ψύξης και αφήνουμε το δίκτυο να ταλαντεύεται μέχρι να φτάσει σε θερμική ισορροπία. Τέλος, αυξάνουμε τα βάρη μεταξύ δύο οποιονδήποτε νευρώνων οι οποίοι είναι ενεργοί. Στη δεύτερη φάση παγιώνουμε μόνο τα νευρώνια εισόδου με το αντίστοιχο πρότυπο εισόδου, αρχικοποιούμε τα νευρώνια εξόδου και τα κρυφά νευρώνια με τυχαίες τιμές και εφαρμόζουμε πάλι την τεχνική της προσομοιωμένης ψύξης. Στο τέλος μειώνουμε τα βάρη μεταξύ δύο οποιονδήποτε νευρώνων οι οποίοι είναι ενεργοί. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να σταθεροποιηθούν τα βάρη. Ο έλεγχος, που ακολουθεί την εκπαίδευση, ενός δικτύου Boltzmann γίνεται ακριβώς με τον ίδιο τρόπο όπως και στα δίκτυα Hopfield. Όταν παρουσιάσουμε, δηλαδή, στο δίκτυο μας ένα καινούριο πρότυπο, τότε το δίκτυο ταλαντεύεται, μέχρι να καταλήξει στο πιο κοντινό ενεργειακά πρότυπο και να το ανακτήσει.

Η αλλαγή στην κατάσταση ενός νευρώνα είναι ανεξάρτητη από την προηγούμενη κατάσταση, όπως και στα δίκτυα Hopfield, γίνεται όμως με βάση τη στοχαστική συνάρτηση κατωφλίου. Δηλαδή, η ίδια διαφορά ενέργειας ΔE_κ ενός συγκεκριμένου νευρώνα, μπορεί είτε να έχει αποτέλεσμα +1 είτε -1. Η διαφορά ενέργειας ΔE_κ και η πιθανότητα P_k που αναλογεί σε κάθε νευρώνα υπολογίζονται με τους αντίστοιχους τύπους:

$$\Delta E_\kappa = \sum_{i=1}^{n} w_{ki} o_i - \theta_k \qquad\qquad P_k = \frac{1}{1+e^{-\Delta E_\kappa / T}}$$

Όπου *n* ο αριθμός των νευρώνων ενός δικτύου, w_{ki} το βάρος μεταξύ των νευρώνων k και i, o_i η έξοδος του νευρώνα i, θ_k το κατώφλι του νευρώνα k και T η θερμοκρασία. Ένας νευρώνας ενεργοποιείται (+1) με πιθανότητα P_k και απενεργοποιείται (-1) με πιθανότητα $(1- P_k)$.

Ένα δίκτυο υλοποιημένο με μηχανές Boltzmann αναμένεται σαφώς να έχει καλύτερα αποτελέσματα απ' ότι με δίκτυα Hopfield. Εντούτοις, υπάρχουν κάποια προβλήματα που σχετίζονται με τις μηχανές Boltzmann, όπως για παράδειγμα πόσο πρέπει να αλλάζουν τα βάρη στο τέλος κάθε φάσης, πως θα προσαρμόζεται η θερμοκρασία κατά τη διάρκεια της προσομοιωμένης ψύξης και πως ξέρουμε αν το δίκτυο έφτασε σε θερμική ισορροπία. Το κυριότερο όμως πρόβλημα των μηχανών Boltzmann είναι ότι χρειάζεται πάρα πολύ χρόνο για να εκπαιδευτεί.

Κεφάλαιο 4

Σχεδιασμός και Υλοποίηση Νευρωνικών Δικτύων

4.1 Προεπεξεργασία δεδομένων εισόδου

Οι εικόνες προσώπου που χρησιμοποιήθηκαν για την εκπαίδευση και τον έλεγχο των δικτύων Hopfield και Boltzmann έπρεπε να επεξεργαστούν και να κωδικοποιηθούν με τον ίδιο ακριβώς τρόπο όπως και στη μελέτη [4].

Όπως αναφέρθηκε και πιο πάνω, κάθε εικόνα προσώπου αποτελείται από 68 συντεταγμένες σημείων. Τα σημεία αυτά, για να μπορούν να χρησιμοποιηθούν ως δεδομένα εισόδου στα νευρωνικά δίκτυα, πρέπει να κωδικοποιηθούν με τέτοιο τρόπο έτσι ώστε για κάθε εικόνα να αντιστοιχεί ένα δυαδικό διάνυσμα. Για να επιτευχθεί η κωδικοποίηση αυτή, κλιμακώνουμε πρώτα το κάθε σημείο (x,y) εντός του διαστήματος [0,31] και ακολούθως μετατρέπουμε την κάθε συντεταγμένη ξεχωριστά σε δυαδική μορφή με 5 bit κωδικοποίηση. Έτσι, με αυτό το τρόπο κάθε εικόνα προσώπου αντιπροσωπεύεται από ένα δυαδικό διάνυσμα με 680 δυαδικά ψηφία (2 x 5 x 68).

Κατά τον έλεγχο των δικτύων, τα πρότυπα που ανακτώνται αποκωδικοποιούνται έτσι ώστε να μετατραπούν ξανά σε συντεταγμένες σημείων. Τα δυαδικά διανύσματα μετατρέπονται αρχικά σε δεκαδική μορφή ανά 5 ψηφία και ακολούθως οι τιμές που αντιστοιχούν στα (x,y) κλιμακώνονται στο αρχικό διάστημα τιμών που είχαν πριν κλιμακωθούν στο διάστημα [0,31]. Έτσι, το δυαδικό διάνυσμα με τα 680 ψηφία μετατρέπεται σε 68 σημεία μορφής (x,y).

Για την υλοποίηση της 5 bit κωδικοποίησης χρειαζόταν να μετατρέψω την κάθε συντεταγμένη σε δυαδική μορφή. Η μετατροπή αυτή όμως προϋποθέτει ότι ο αριθμός που θα μετατραπεί είναι ακέραιος, ενώ τα σημεία είναι σε δεκαδική μορφή. Χρειάστηκε, λοιπόν, να αφαιρέσω το δεκαδικό μέρος από κάθε συντεταγμένη και να χρησιμοποιήσω μόνο το ακέραιο. Παρατηρώντας το σχήμα 2.2 βλέπουμε ότι τα σημεία δεν έχουν τόσο μικρή απόσταση μεταξύ τους, ούτως ώστε να επηρεάσει τόσο πολύ τα αποτελέσματα η αποκοπή του δεκαδικού μέρους κάθε σημείου. Τα ανακτηθέντα σημεία, επίσης αποτελούνται μόνο από ακέραιο μέρος.

Λόγω του προβλήματος που αντιμετωπίζουν τα δίκτυα Hopfield με τη χωρητικότητα τους, πρέπει να χρησιμοποιηθούν μόνο τα 102 πρότυπα από τα 1002 για την εκπαίδευση του δικτύου, εφόσον το δίκτυο αποτελείται από 680 νευρώνες (15% των 680 νευρώνων). Ο περιορισμός αυτός ισχύει και για τις μηχανές Boltzmann, αφού θέλουμε να συγκρίνουμε τα αποτελέσματα των δύο δικτύων με τα ίδια δεδομένα εισόδου.

4.2 Στατιστικές μέθοδοι για υπολογισμό σφάλματος

Για τον υπολογισμό του σφάλματος αλλά και για την σύγκριση απόδοσης των δύο νευρωνικών δικτύων χρησιμοποιήσαμε δύο μετρικές σφάλματος, το ολικό σφάλμα (overall error) και το περιορισμένο σφάλμα (restricted error). Το ολικό σφάλμα είναι η μέση Ευκλείδια απόσταση μεταξύ των αρχικών εικόνων προσώπων και των ανακτηθέντων εικόνων. Το περιορισμένο σφάλμα είναι η μέση Ευκλείδια απόσταση μεταξύ των αρχικών σημείων των επικαλυμμένων μερών μιας εικόνας προσώπου και των ανακτηθέντων σημείων. Ο τύπος για τις δύο αυτές μετρήσεις είναι ο εξής:

$$\text{Error(n)}= \frac{1}{n}\sqrt{\sum_{i=1}^{n}((x_2\text{-}x_1)^2+(y_2\text{-}y_1)^2)}$$

όπου (x_1, y_1) το αρχικό σημείο, (x_2, y_2) το ανακτηθέν σημείο και n ο αριθμός των σημείων μιας εικόνας προσώπου (68 σημεία).

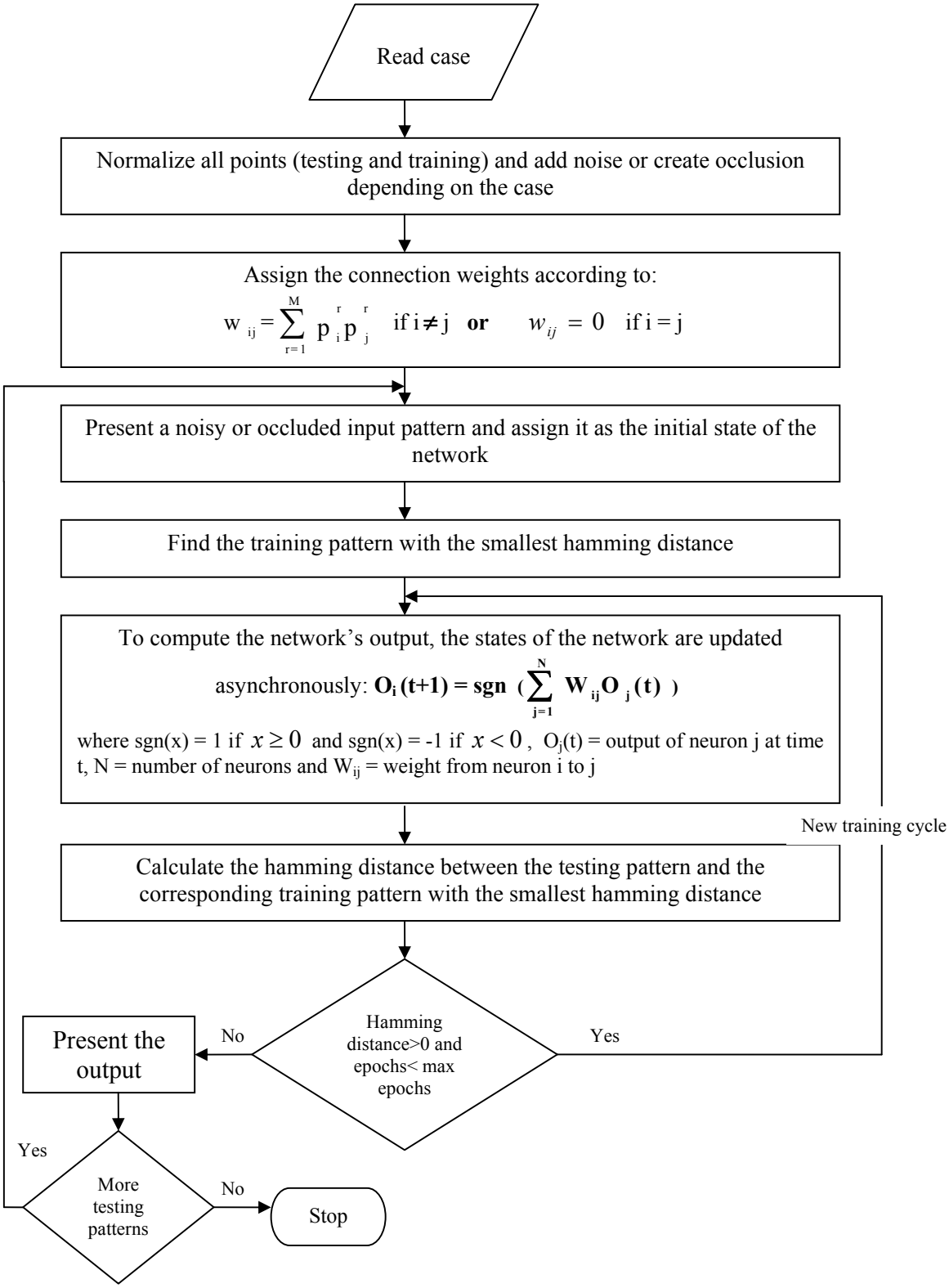
Επίσης, υπολογίσαμε την τυπική απόκλιση για το ολικό σφάλμα, καθώς και για το περιορισμένο σφάλμα. Ο τύπος για την τυπική απόκλιση που χρησιμοποιήσαμε είναι ο εξής:

$$\text{Standard Deviation} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d(n) - d\_mean)}$$

Όπου n ο αριθμός των εικόνων προσώπου που χρησιμοποιούνται για τον έλεγχο των δικτύων, $d(n) = \frac{1}{k}\sum_{i=1}^{k}\sqrt{(x_2\text{-}x_1)^2+(y_2\text{-}y_1)^2}$, $d\_mean = \frac{1}{n}\sum_{i=1}^{n} d(n)$ και k ο αριθμός των σημείων.

4.3 Αλγόριθμος Δικτύου Hopfield

Στο σχήμα 4.1 παρουσιάζεται ο γενικός αλγόριθμος που χρησιμοποιήθηκε για την υλοποίηση του δικτύου Hopfield σε μορφή διαγράμματος ροής.



Σχήμα 4.1: Αλγόριθμος υλοποίησης δικτύου Hopfield

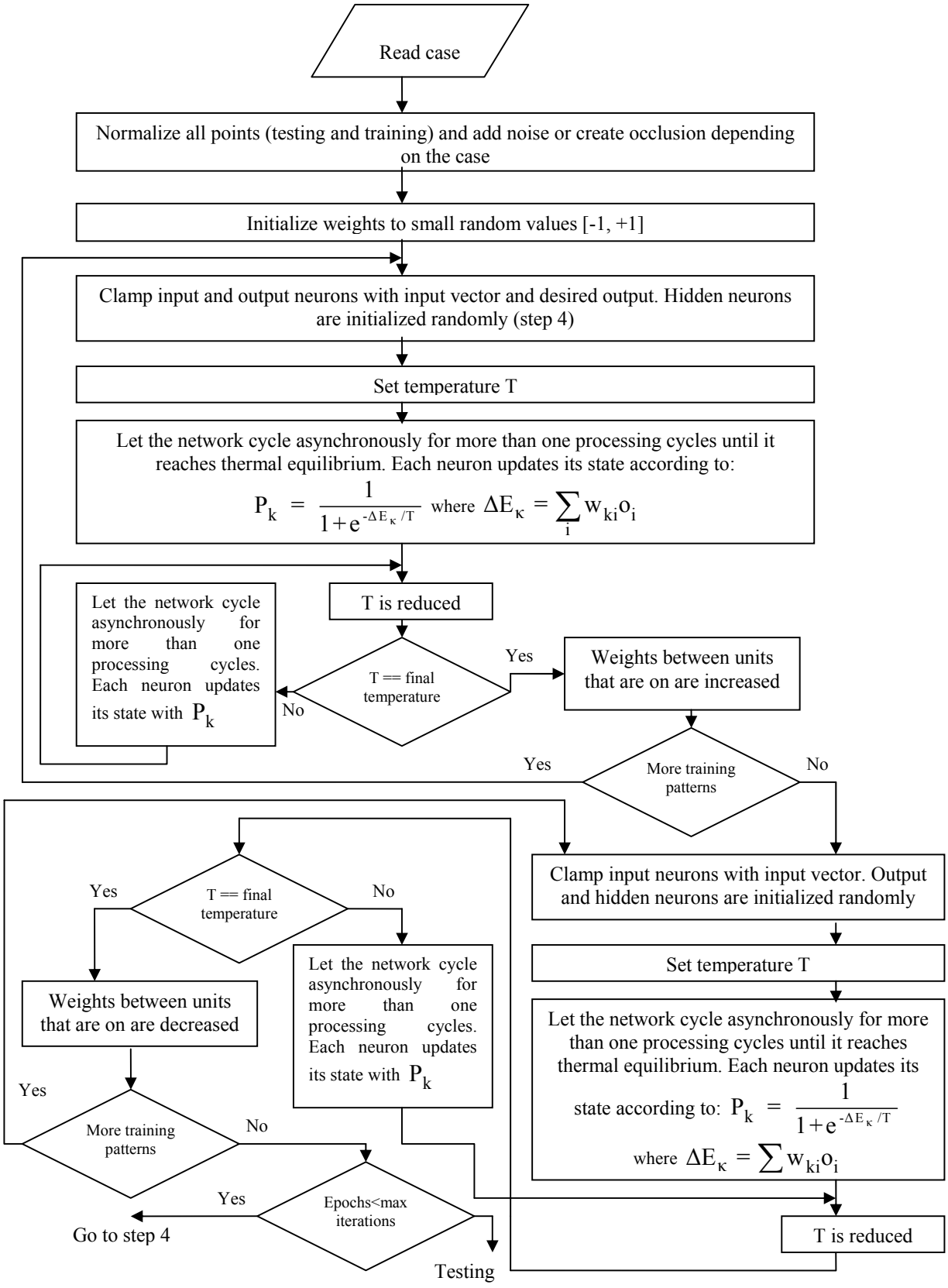
Στο πρώτο βήμα καθορίζεται η περίπτωση από τις δύο κατηγορίες που φαίνονται στα σχήματα 2.3 και 2.4, λόγω του ότι στο πρόγραμμα μια περίπτωση μπορεί να τρέχει κάθε φορά. Η περίπτωση διαβάζεται από ένα αρχείο. Ακολούθως, διαβάζονται όλα τα δεδομένα εισόδου, αυτά που θα χρησιμοποιηθούν στην εκπαίδευση αλλά και στον έλεγχο και κανονικοποιούνται, όπως ακριβώς περιγράψαμε στο υποκεφάλαιο 4.1.

Η ανάθεση των βαρών μεταξύ των νευρώνων, στο βήμα 3, είναι πολύ σημαντική γιατί αυτή μας εξασφαλίζει ότι το δίκτυο μας θα καταλήξει σε ελάχιστο σημείο. Στο βήμα 4 παρουσιάζουμε στο δίκτυο ένα θορυβώδες ή επικαλυμμένο πρότυπο, από αυτά που κανονικοποιήθηκαν στο βήμα 2 και υπολογίζουμε ποιο πρότυπο εισόδου έχει το μικρότερο hamming distance με το συγκεκριμένο πρότυπο. Στη συνέχεια, αφήνουμε το δίκτυο να ταλαντωθεί μέχρι να γίνει το hamming distance ίσο με μηδέν ή μέχρι οι επαναλήψεις να φτάσουν το μέγιστο αριθμό. Μετά από πολλές δοκιμές καταλήξαμε στο ότι ο βέλτιστος αριθμός επαναλήψεων πρέπει να είναι 3000. Όταν σταματήσει το δίκτυο να ταλαντεύεται, τότε το πρότυπο που έχει ανακτηθεί μετατρέπεται σε συντεταγμένες σημείων. Η διαδικασία που ακολουθεί την ανάθεση βαρών (η ανάθεση βαρών γίνεται μόνο μια φορά) επαναλαμβάνεται για όλα τα θορυβώδη ή επικαλυμμένα πρότυπα, στην περίπτωση μας και για τα 900 πρόσωπα.

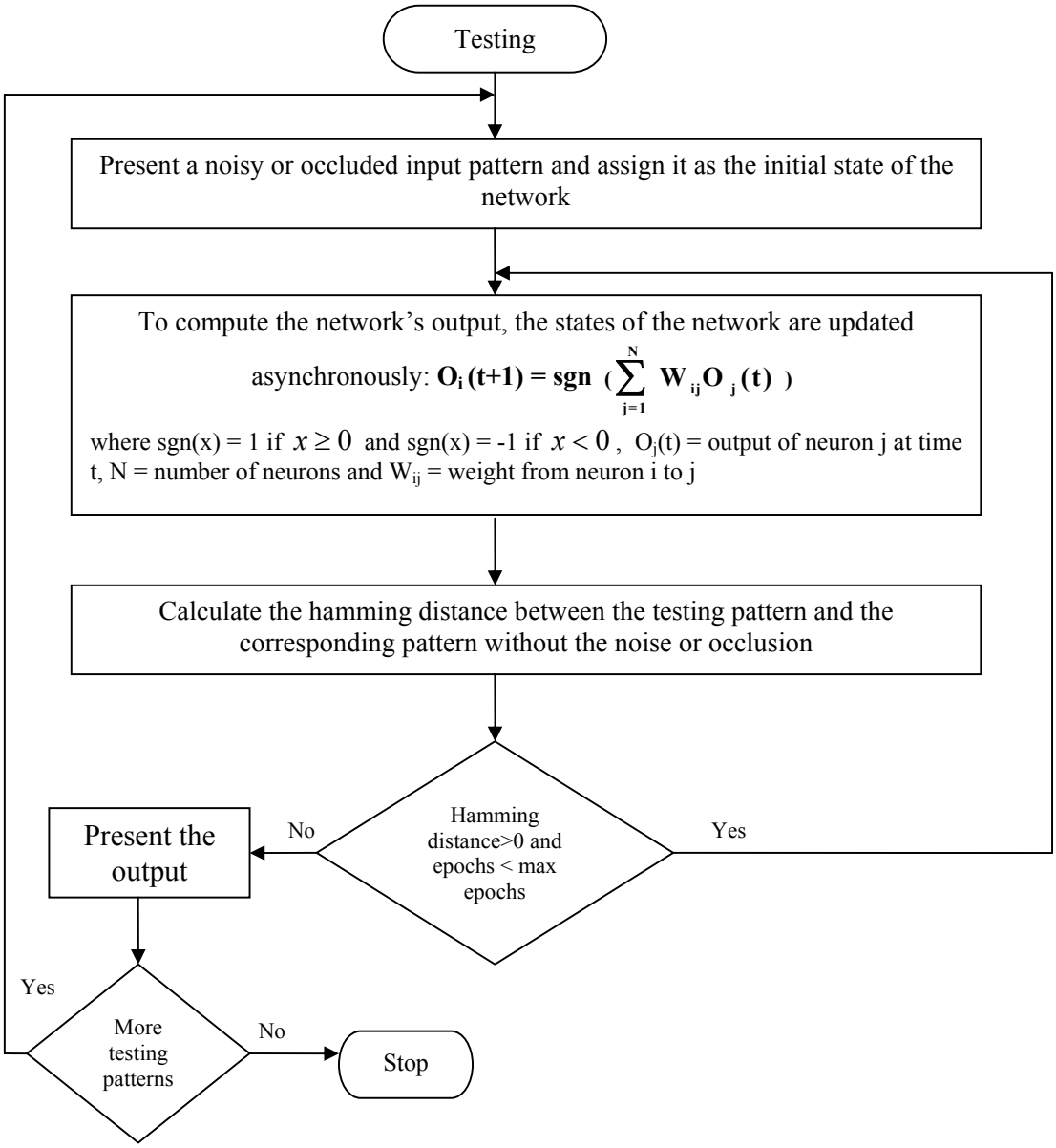
Τέλος, υπολογίζουμε τις μετρικές σφάλματος που περιγράψαμε στο υποκεφάλαιο 4.2. Για τον υπολογισμό του ολικού σφάλματος, αρχικά υπολογίζουμε την μέση Ευκλείδια απόσταση για κάθε πρόσωπο, αθροίζουμε τις αποστάσεις αυτές και για τα 900 πρότυπα και ακολούθως βρίσκουμε το μέσο σφάλμα για όλα τα πρόσωπα. Έτσι, έχουμε για κάθε μια από τις περιπτώσεις (1-11) ένα μέσο σφάλμα. Για τον υπολογισμό του περιορισμένου σφάλματος, η διαφορά έγκειται στο ότι για κάθε πρόσωπο υπολογίζουμε την Ευκλείδια απόσταση μεταξύ των επικαλυμμένων σημείων μόνο και όχι όλων των σημείων. Το περιορισμένο σφάλμα και η αντίστοιχη τυπική απόκλιση υπολογίζονται μόνο στις περιπτώσεις 1-6, στις οποίες έχουμε επικάλυψη σημείων. Αντίθετα, το ολικό σφάλμα και η αντίστοιχη τυπική απόκλιση υπολογίζονται σε όλες τις περιπτώσεις.

4.4 Αλγόριθμος Μηχανών Boltzmann

Στα σχήματα 4.2 και 4.3 παρουσιάζεται ο αλγόριθμος με βάση τον οποίο υλοποιήθηκε το δίκτυο Boltzmann. Στο σχήμα 4.2 είναι ο αλγόριθμος για την εκπαίδευση του δικτύου και στο σχήμα 4.3 ο αλγόριθμος για τον έλεγχο του δικτύου.



Σχήμα 4.2: Αλγόριθμος εκπαίδευσης δικτύου Boltzmann



Σχήμα 4.3: Αλγόριθμος ελέγχου δικτύου Boltzmann

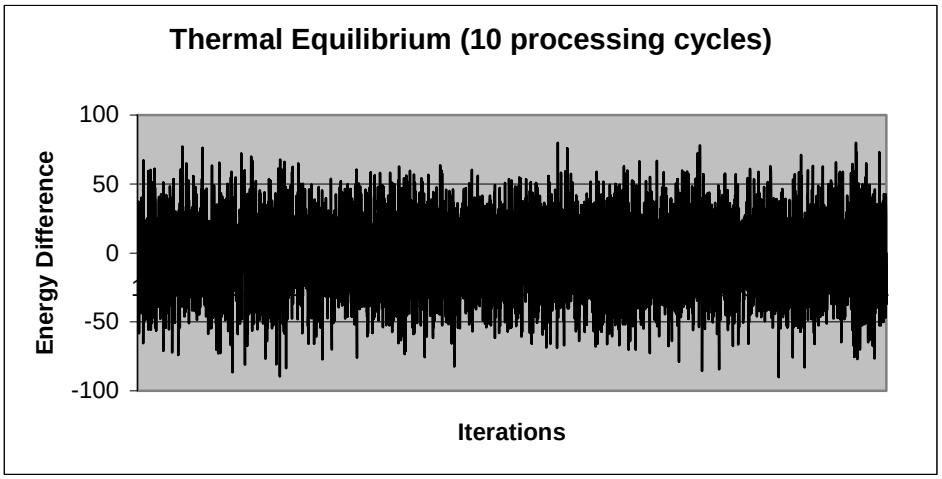
Στην εκπαίδευση του δικτύου Boltzmann τα πρώτα δύο βήματα είναι τα ίδια με αυτά στον αλγόριθμο υλοποίησης Hopfield. Καθορίζεται η περίπτωση (1-11) και κανονικοποιούνται όλα τα δεδομένα που θα χρησιμοποιηθούν είτε για εκπαίδευση είτε για έλεγχο. Ακολουθεί η πρώτη φάση εκπαίδευσης του δικτύου Boltzmann κατά την οποία παγιώνονται τα νευρώνια εισόδου, με το πρότυπο το οποίο πρόκειται να «μάθει»

το δίκτυο και τα νευρώνια εξόδου, με το επιθυμητό αποτέλεσμα. Το επιθυμητό αποτέλεσμα στην περίπτωση της εκπαίδευσης είναι το ίδιο το πρότυπο που παγιώνουμε τα νευρώνια εισόδου. Τα κρυφά νευρώνια αρχικοποιούνται τυχαία είτε με +1 είτε με -1. Ο αριθμός των νευρώνων εισόδου και εξόδου είναι 680, όσα και τα νευρώνια που αποτελούν το δίκτυο μας, ενώ ο αριθμός των κρυφών νευρώνων είναι 700.

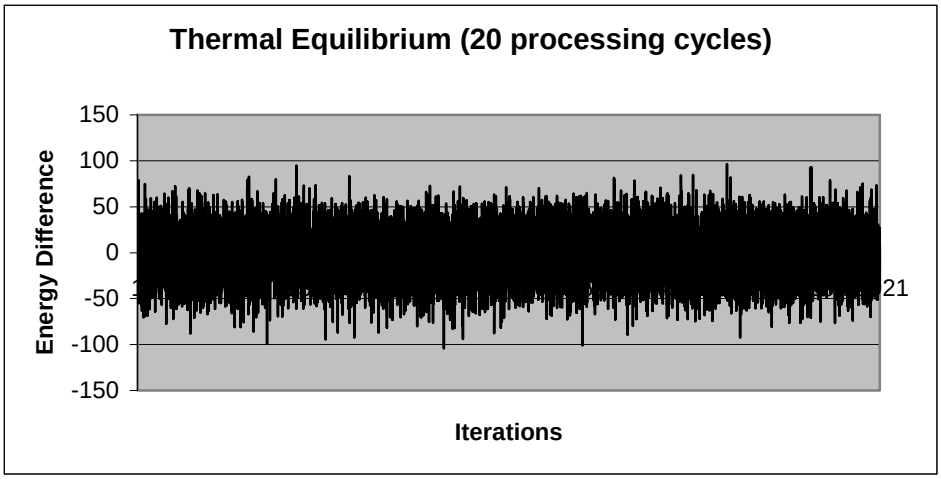
Αφού παγιώσουμε τα νευρώνια εισόδου και εξόδου, εφαρμόζεται η μέθοδος της προσομοιωμένης ψύξης. Μετά από αρκετές δοκιμές, θέσαμε την αρχική θερμοκρασία του δικτύου 20 και την τελική θερμοκρασία 0.005. Αρχικά, δοκιμάσαμε με πιο ψηλές αρχικές θερμοκρασίες, για παράδειγμα 100, όμως ήταν αχρείαστο αφού το σύστημα έφτανε σε θερμική ισορροπία ξεκινώντας και από το 20. Αφήνουμε το δίκτυο να ταλαντωθεί με βάση τους τύπους $P_k = \frac{1}{1+e^{-\Delta E_\kappa / T}}$ και $\Delta E_\kappa = \sum_i w_{ki} o_i$, με την αρχική θερμοκρασία να ισούται με 20, μέχρι να φτάσει σε θερμική ισορροπία. Όταν το δίκτυο φτάσει σε θερμική ισορροπία, τότε όλα τα νευρώνια συνεχίζουν να αλλάζουν καταστάσεις με σταθερές όμως πιθανότητες. Ακολούθως, μειώνουμε τη θερμοκρασία και αφήνουμε πάλι το δίκτυο να ταλαντωθεί για όσες επαναλήψεις χρειάστηκε για να φτάσει σε θερμική ισορροπία. Κάθε φορά η θερμοκρασία μειώνεται μέχρι να φτάσει στην τελική της τιμή 0.005 σύμφωνα με τον τύπο: $T = T * a$, όπου το *α* τέθηκε σαν 0.99.

Στην πρώτη φάση της εκπαίδευσης του δικτύου Boltzmann, ταλαντεύονται μόνο τα κρυφά νευρώνια (σύνολο 700) και κάθε φορά επιλέγεται τυχαία ποιο νευρώνιο θα αλλάξει κατάσταση. Αν αφήσουμε το δίκτυο να ταλαντωθεί μόνο για 700 επαναλήψεις, όσες και τα νευρώνια που πρέπει να αλλάξουν κατάσταση, δεν υπάρχει εγγύηση ότι όλα τα νευρώνια θα αλλάξουν κατάσταση, εφόσον επιλέγονται τυχαία. Πρέπει να αυξήσουμε λοιπόν τον αριθμό των processing cycles, ένα processing cycle είναι όταν αφήνουμε ένα δίκτυο με **n** νευρώνια που πρέπει να αλλάξουν κατάσταση να τρέξει για **n** επαναλήψεις. Δοκιμάσαμε να τρέξουμε το πρόγραμμα με 10 και 20 processing cycles για να δούμε αν το δίκτυο φτάνει σε θερμική ισορροπία. Στις γραφικές παραστάσεις 4.4 και 4.5 βλέπουμε αν το δίκτυο φτάνει σε θερμική ισορροπία βάση του πως μεταβάλλεται η διαφορά ενέργειας ($\Delta E_\kappa = \sum_i w_{ki} o_i$) ανάλογα με τις επαναλήψεις. Σύμφωνα με τη θεωρία όταν

το δίκτυο είναι σε θερμική ισορροπία και η θερμοκρασία στη μέγιστη της τιμή, το δίκτυο εκτελεί τις μέγιστες του ταλαντώσεις. Άρα, μεταβαίνει συνεχώς από αρνητικές σε θετικές καταστάσεις και αντίθετα. Στις γραφικές παραστάσεις, φαίνεται αυτή η έντονη εναλλαγή από θετικές σε αρνητικές τιμές και αντίθετα και έτσι συμπεραίνουμε ότι το δίκτυο μας φτάνει σε θερμική ισορροπία και με 10 και με 20 processing cycles. Για θέμα χρόνου όμως προτιμήσαμε 10 processing cycles. Στη δεύτερη φάση της εκπαίδευσης ταλαντεύονται εκτός από τα κρυφά νευρώνια και τα νευρώνια εξόδου (σύνολο 1380). Ο αριθμός των processing cycles παραμένει 10.



Σχήμα 4.4: Γραφική παράσταση που δείχνει πως το δίκτυο φτάνει σε θερμική ισορροπία με 10 processing cycles



Σχήμα 4.5: Γραφική παράσταση που δείχνει πως το δίκτυο φτάνει σε θερμική ισορροπία με 20 processing cycles

Όταν τελειώσει η πρώτη φάση της εκπαίδευσης, όταν δηλαδή η θερμοκρασία πάρει την τελική της τιμή, τότε αυξάνουμε τα βάρη μεταξύ δύο οποιονδήποτε ενεργών νευρώνων κατά 0.005. Επιλέγουμε 0.005 που είναι μια μικρή τιμή για να αποφύγουμε υπερβολικές αυξήσεις στα βάρη. Στη συνέχεια, επαναλαμβάνουμε όλη την πιο πάνω διαδικασία για όλα τα πρότυπα που θα χρησιμοποιηθούν στην εκπαίδευση.

Στη δεύτερη φάση, παγιώνουμε μόνο τα νευρώνια εισόδου και αρχικοποιούμε τυχαία τα νευρώνια εξόδου και τα κρυφά νευρώνια. Η προσομοιωμένη ψύξη εφαρμόζεται ξανά όπως και στη πρώτη φάση, αφού φτάσει το δίκτυο μας σε θερμική ισορροπία με τη θερμοκρασία στη μέγιστη της τιμή, με τις ίδιες θερμοκρασίες και τον ίδιο αριθμό processing cycles. Όταν τελειώσει η δεύτερη φάση μειώνουμε τα βάρη μεταξύ δύο οποιονδήποτε ενεργών νευρώνων κατά 0.005 και επαναλαμβάνουμε τη δέυτερη φάση για όλα τα 102 πρότυπα. Η διαδικασία της εκπαίδευσης επαναλαμβάνεται για 8 επαναλήψεις. Λόγω έλλειψης χρόνου δεν μπορέσαμε να βάλουμε περισσότερες επαναλήψεις.

Ο έλεγχος του δικτύου Boltzmann, όπως φαίνεται και στο σχήμα 4.3 είναι ίδιος με τον έλεγχο στο δίκτυο Hopfield. Η διαφορά είναι ότι σε κάθε επανάληψη υπολογίζουμε το hamming distance μεταξύ του θορυβώδους ή επικαλυμμένου πρότυπου με το ίδιο

πρότυπο προτού προσθέσουμε θόρυβο ή το επικαλύψουμε. Ο μέγιστος αριθμός επαναλήψεων είναι επίσης 3000, όπως και στο Hopfield. Μετά τον έλεγχο, εφαρμόζουμε τις μετρικές σφάλματος για να εξετάσουμε τα αποτελέσματα του δικτύου Boltzmann.

Κεφάλαιο 5

Αποτελέσματα και Συζήτηση

5.1 Αποτελέσματα Δικτύου Hopfield

Στους πίνακες 5.1 και 5.2 παρουσιάζονται τα αποτελέσματα του δικτύου Hopfield για τις διάφορες περιπτώσεις 1-11 που περιγράψαμε στο υποκεφάλαιο 2.1 (σχήματα 2.3 και 2.4). Στον πίνακα 5.1 φαίνονται οι μετρήσεις για το ολικό και το περιορισμένο σφάλμα, καθώς και η τιμή της τυπικής απόκλισης που αντιστοιχεί στον κάθε τύπο σφάλματος, για τις περιπτώσεις 1-6. Στον πίνακα 5.2 γίνεται αναφορά για τις υπόλοιπες περιπτώσεις (1-11) και φαίνονται τα αποτελέσματα για το ολικό σφάλμα και την τυπική απόκλιση.

Metrics	Case Number						Average of means
	1	2	3	4	5	6	
Overall Mean	1.31	1.49	1.51	1.89	1.64	1.62	1.57
Overall st. deviation	1.61	1.76	1.76	9.14	3.49	1.84	
Restricted Mean	4.09	6.73	6.49	9.19	7.47	7.75	6.95
Restricted st. deviation	5.61	8.61	3.46	6.48	8.92	4.05	

Πίνακας 5.1: Σφάλμα και τυπική απόκλιση της Ευκλείδιας απόστασης μεταξύ των αρχικών και των ανακτηθέντων εικόνων προσώπου για τις περιπτώσεις 1-6. Οι περιπτώσεις 1-6 αφορούν εικόνες με επικάλυψη

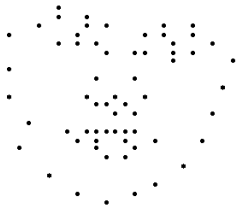
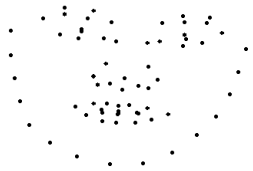
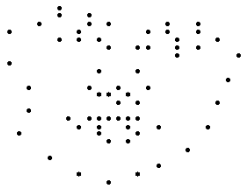
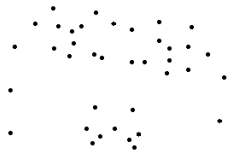
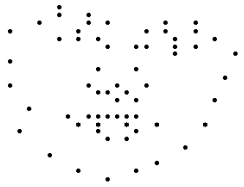
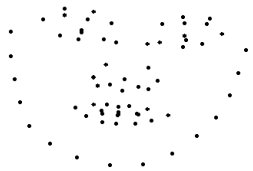
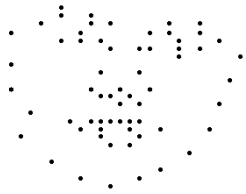
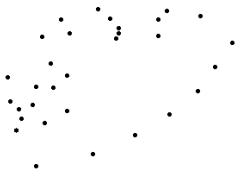
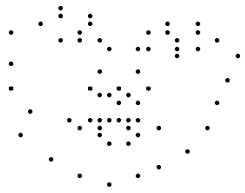
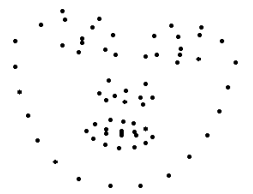
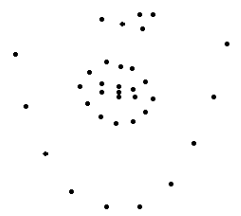
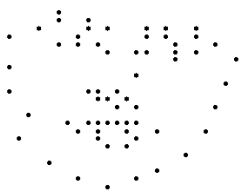
Στον πίνακα 5.1 φαίνονται τα αποτελέσματα από τον έλεγχο του δικτύου Hopfield με εικόνες που έχουν μερικώς επικαλυφθεί. Σύμφωνα με τα πιο πάνω αποτελέσματα παρατηρούμε, όπως ήταν αναμενόμενο, ότι όσο πιο μεγάλο μέρος επικαλύπτεται σε μια εικόνα προσώπου το ποσοστό σφάλματος είναι μεγάλο και μειώνεται σταδιακά όταν επικαλύπτονται λιγότερα σημεία της εικόνας προσώπου. Στην περίπτωση 4, όπως φαίνεται και στο σχήμα 2.3, έχουμε το μεγαλύτερο μέρος της εικόνας επικαλυμμένο, γι'αυτό βλέπουμε το ολικό και το περιορισμένο σφάλμα να παίρνουν τις μέγιστες τους τιμές στη συγκεκριμένη περίπτωση. Αντιθέτως, στην περίπτωση 1 που επικαλύπτεται μόνο ένα πολύ μικρό μέρος της εικόνας, το ολικό και περιορισμένο σφάλμα έχουν χαμηλές τιμές.

Metrics	Case Number					Average of means
	7	8	9	10	11	
Overall Mean	1.25	1.37	1.47	1.60	1.63	1.46
Overall st. deviation	1.64	1.62	1.78	1.87	1.83	

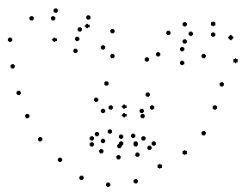
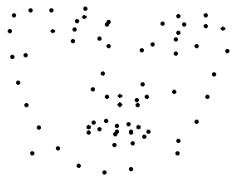
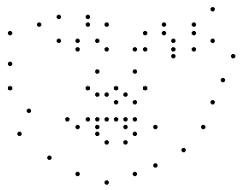
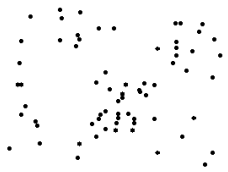
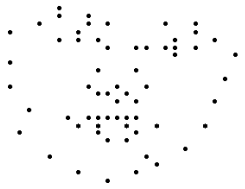
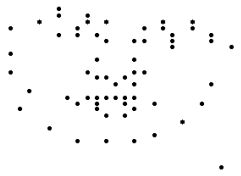
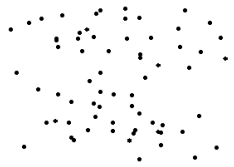
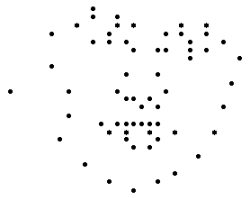
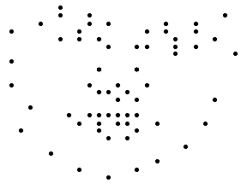
Πίνακας 5.2: Σφάλμα και τυπική απόκλιση της Ευκλείδιας απόστασης μεταξύ των αρχικών και των ανακτηθέντων εικόνων προσώπου για τις περιπτώσεις 7-11. Οι περιπτώσεις 7-11 αφορούν εικόνες με θόρυβο

Στον πίνακα 5.2 παρουσιάζονται τα αποτελέσματα για τις περιπτώσεις 7-11. Σε αυτές τις περιπτώσεις οι εικόνες προσώπου που χρησιμοποιήθηκαν για τον έλεγχο του δικτύου ήταν θορυβώδεις. Βάσει των αποτελεσμάτων διαπιστώνουμε ότι όσο προστίθεται στην εικόνα περισσότερος θόρυβος (10%, 30%, 50%, 70%, 90% των σημείων), τόσο αυξάνεται το σφάλμα.

Η απόδοση του δικτύου Hopfield φαίνεται, επίσης, στα σχήματα 5.1 και 5.2 με την αναπαράσταση των εικόνων προσώπου με 68 σημεία. Για κάθε περίπτωση (1-11) φαίνεται η αρχική εικόνα προσώπου, η επικαλυμμένη ή θορυβώδης εικόνα που παρουσιάζεται στο δίκτυο για έλεγχο και η ανακτημένη.

Case	Original Image	Occluded Image	Retrieved Image
Case 1			
Case 2			
Case 3			
Case 4			
Case 5			
Case 6			

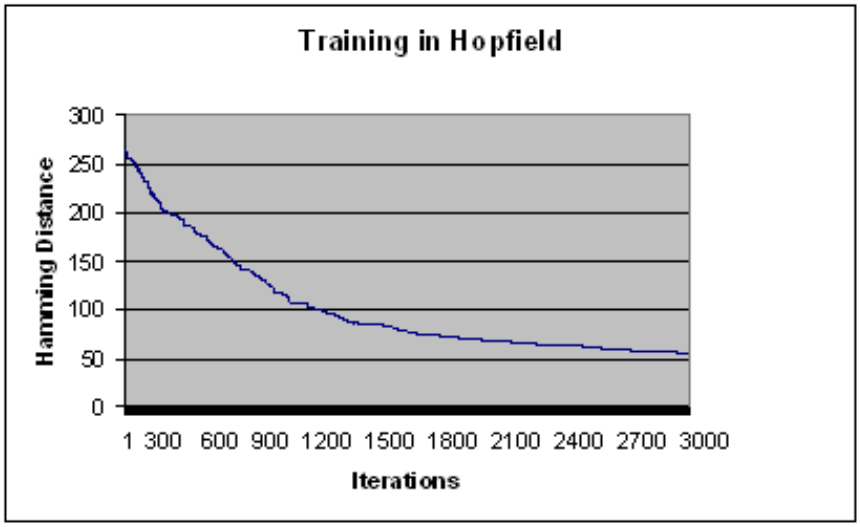
Σχήμα 5.1: Αρχικές, επικαλυμμένες και ανακτημένες εικόνες προσώπου αναπαριστάμενες με 68 σημεία. Περιπτώσεις επικάλυψης 1-6

Case	Original Image	Occluded Image	Retrieved Image
Case 7			
Case 8			
Case 9			
Case 10			
Case 11			

Σχήμα 5.2: Αρχικές, θορυβώδεις και ανακτημένες εικόνες προσώπου αναπαριστάμενες με 68 σημεία. Περιπτώσεις θορύβου 7-11

Για να διαπιστώσουμε ότι η εκπαίδευση του δικτύου Hopfield είναι επιτυχής, παρατηρούμε αν και κατά πόσο μειώνεται το Hamming Distance σε κάθε επανάληψη κατά τη διάρκεια της εκπαίδευσης. Κάθε φορά που παρουσιάζουμε στο δίκτυο ένα θορυβώδες ή επικαλυμμένο πρότυπο, υπολογίζουμε το μικρότερο hamming distance μεταξύ του συγκεκριμένου πρότυπου και των προτύπων που προορίζονται για την

εκπαίδευση. Ακολούθως, σε κάθε επανάληψη που ταλαντεύεται το δίκτυο υπολογίζουμε το hamming distance μεταξύ του θορυβώδους ή του επικαλυμμένου πρότυπου, το οποίο αλλάζει κάθε φορά και του αντίστοιχου προτύπου με το μικρότερο hamming distance. Αν το hamming distance μειώνεται σημαίνει ότι το δίκτυο «μαθαίνει». Στο σχήμα 5.3 φαίνεται πως μειώνεται το hamming distance με 3000 επαναλήψεις.



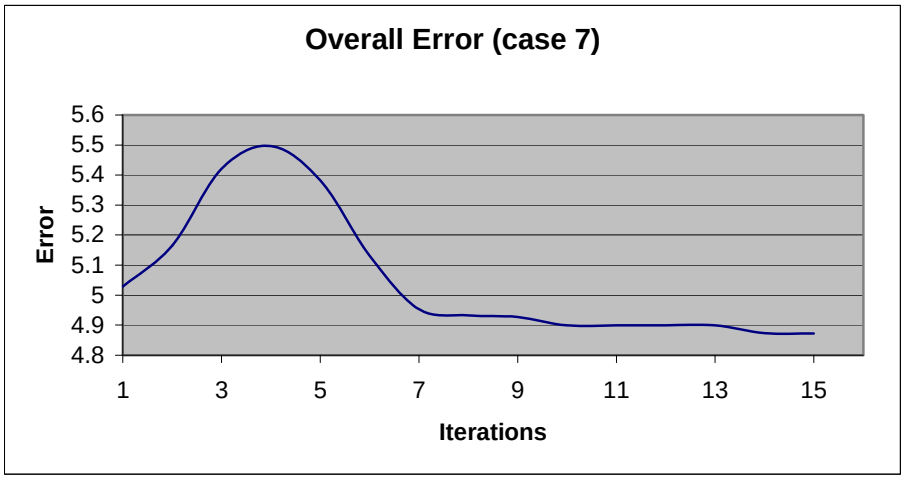
Σχήμα 5.3: Γραφική παράσταση που δείχνει τη μείωση του Hamming Distance κατά τη διάρκεια της εκπαίδευσης του δικτύου Hopfield

5.2 Αποτελέσματα Μηχανών Boltzmann

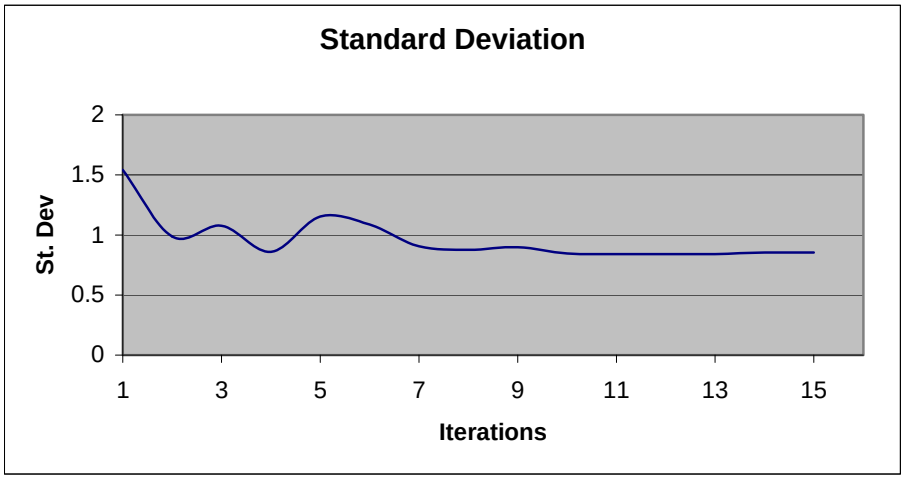
Η υλοποίηση του δικτύου Boltzmann έχει γίνει, χωρίς όμως τα αναμενόμενα αποτελέσματα. Ο χρόνος που είχε απομείνει για την υλοποίηση του δικτύου ήταν πολύ λίγος. Οι κύριοι λόγοι είναι γιατί η υλοποίηση του δικτύου Hopfield είχε καθυστερήσει να ολοκληρωθεί και γιατί η προσομοίωση του δικτύου Boltzmann είναι εξαιρετικά αργή. Σε 24 ώρες ολοκληρώνονταν σχεδόν δύο μόνο επαναλήψεις. Ο μέγιστος αριθμός επαναλήψεων που είχε τεθεί, ούτως ώστε να έχουμε αποτελέσματα εντός χρονικών πλαισίων, ήταν 15. Όμως, δεκαπέντε επαναλήψεις είναι πάρα πολύ λίγες για να εκπαιδευτεί το δίκτυο Boltzmann και γι'αυτό δεν είχαμε τα επιθυμητά αποτελέσματα.

Ακόμη μια συνέπεια της έλλειψης χρόνου, αλλά και πιθανή αιτία που τα αποτελέσματα δεν ήταν τα αναμενόμενα, ήταν το ότι δεν πειραματιστήκαμε με διάφορες παραμέτρους που είχαμε αρχικοποιήσει με σχεδόν τυχαίες τιμές. Όπως για παράδειγμα, ο αριθμός των κρυφών νευρώνων. Είχαμε θέσει 700 κρυφά νευρώνια, ένα αριθμό λίγο μεγαλύτερο από τον αριθμό των νευρώνων εισόδου και εξόδου, που είναι 680. Ίσως, όμως, χρειάζεται ο αριθμός των κρυφών νευρώνων να είναι μεγαλύτερος για καλύτερη κωδικοποίηση του δικτύου. Επίσης, τα βάρη στο τέλος κάθε φάσης, κατά τη διάρκεια της εκπαίδευσης, αυξάνονταν ή μειώνονταν κατά 0.005. Η τιμή αυτή είναι τυχαία και επιλέχθηκε τόσο μικρή για να μην έχουμε υπερβολικές αυξήσεις στα βάρη.

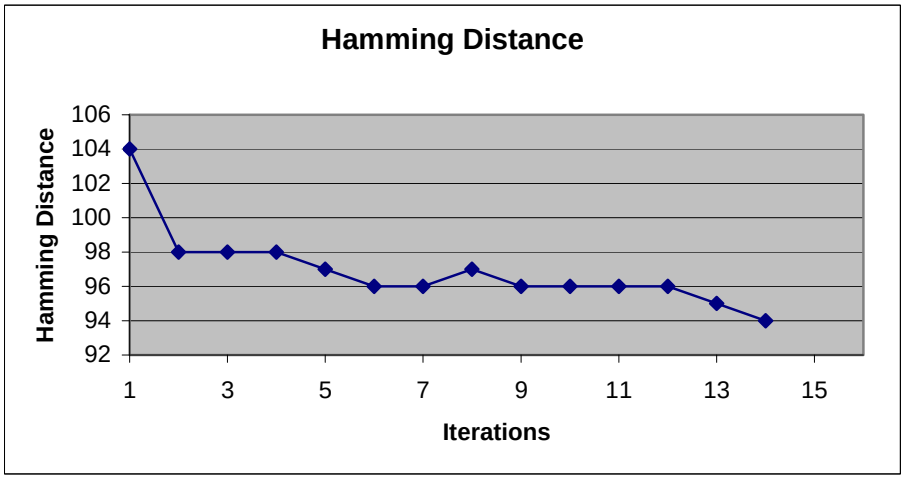
Στις γραφικές παραστάσεις 5.4 και 5.5 φαίνεται πως μεταβάλλεται το ολικό σφάλμα και η αντίστοιχη τυπική απόκλιση, ανάλογα με τον αριθμό των επαναλήψεων. Η μόνη περίπτωση που έχει υλοποιηθεί, από αυτές που φαίνονται στα σχήματα 2.3 και 2.4, είναι η περίπτωση 7. Η μείωση του σφάλματος που παρατηρείται στην πιο κάτω γραφική παράσταση, φανερώνει ότι το δίκτυο Boltzmann εκπαιδεύεται σωστά. Οι ψηλές τιμές πιθανόν να οφείλονται στο μικρό αριθμό επαναλήψεων. Επίσης, ελέγξαμε το hamming distance και παρατηρήσαμε ότι μειώνεται με την αύξηση των επαναλήψεων. Η γραφική παράσταση 5.6 παρουσιάζει τη μείωση αυτή.



Σχήμα 5.4: Γραφική παράσταση που απεικονίζει τις τιμές του ολικού σφάλματος στην περίπτωση 7



Σχήμα 5.5: Γραφική παράσταση που απεικονίζει τις τιμές της τυπικής απόκλισης που αντιστοιχούν στο ολικό σφάλμα της γραφικής παράστασης 5.4



Σχήμα 5.6: Γραφική παράσταση που απεικονίζει τη μείωση του hamming distance ανάλογα με την αύξηση των επαναλήψεων

Μελετώντας τα αποτελέσματα που προέκυψαν από τους υπολογισμούς του δικτύου Boltzmann, συμπεραίνουμε ότι ο αρχικός στόχος υλοποίησης δικτύου Boltzmann που να επιλύει το πρόβλημα αποκατάστασης μερικώς επικαλυμμένων εικόνων προσώπου, έχει επιτευχθεί. Το δίκτυο εκπαιδεύεται σωστά, όπως διαπιστώνεται από την μείωση του σφάλματος και του hamming distance αναλόγως της αύξησης των επαναλήψεων, απλά χρειαζόταν ακόμα περισσότερες επαναλήψεις για να ολοκληρωθεί η εκπαίδευση του δικτύου.

5.3 Σύγκριση με προηγούμενη μελέτη

Στο υποκεφάλαιο αυτό θα συγκρίνουμε τα αποτελέσματα του δικτύου Hopfield με τα αποτελέσματα της μελέτης που έχουν υλοποιήσει οι Draganova et al. Στη συγκεκριμένη μελέτη [4] έχει υλοποιηθεί το ίδιο πρόβλημα με δίκτυα Hopfield, με δίκτυα πολλαπλών στρωμάτων Perceptron (MLP), με μια καινούρια τεχνική που συνδυάζει δίκτυα Hopfield και MLP και με μια μέθοδο που βασίζεται σε συσχετιζόμενη αναζήτηση (associative search). Στους πίνακες 5.3, 5.4 και 5.5 παρουσιάζονται τα αποτελέσματα και των δύο δικτύων για όλες τις περιπτώσεις 1-11. Τα αποτελέσματα του δικτύου που υλοποίησαν οι Draganova et al βρίσκονται κάτω από τη κατηγορία H1, ενώ τα αποτελέσματα της παρούσας διπλωματικής βρίσκονται κάτω από τη κατηγορία H2. Οι διαφορές που παρουσιάζονται μεταξύ των αποτελεσμάτων οφείλεται κυρίως στο ότι χρησιμοποιήθηκαν διαφορετικά σημεία για επικάλυψη και για θόρυβο στον έλεγχο της κάθε εικόνας προσώπου.

Metrics	**Case Number**											
	1		**2**		**3**		**4**		**5**		**6**	
	H1	**H2**	**H1**	**H2**	**H1**	**H2**	**H1**	**H2**	**H1**	**H2**	**H1**	**H2**
Overall Mean	0.89	1.31	3.21	**1.49**	4.09	**1.51**	6.11	**1.89**	2.69	**1.64**	2.33	**1.62**
Overall st. deviation	2.33	**1.61**	1.68	1.76	1.77	**1.76**	1.9	9.14	1.72	3.49	1.93	**1.84**
Restricted Mean	6.03	**4.09**	5.90	6.73	6.63	**6.49**	7.16	9.19	5.89	7.47	6.09	7.75
Restricted st. deviation	0.34	5.61	0.91	8.61	1.09	3.46	1.62	6.48	0.78	8.92	0.73	4.05

Πίνακας 5.3: Σύγκριση αποτελεσμάτων της παρούσας διπλωματικής (H2) και της μελέτης των Draganova et al (H1). Το δίκτυο ελέγχθηκε με εικόνες επικαλυμμένες (περιπτώσεις 1-6)

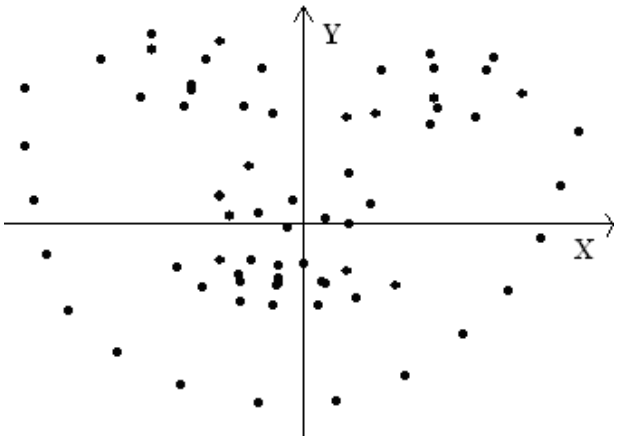
Metrics	Case Number									
	7		8		9		10		11	
	H1	H2	H1	H2	H1	H2	H1	H2	H1	H2
Overall Mean	0.48	1.25	1.34	1.37	2.19	**1.47**	3.19	**1.60**	3.59	**1.63**
Overall st. deviation	2.25	**1.64**	1.77	**1.62**	1.81	**1.78**	1.76	1.87	1.85	**1.83**

Πίνακας 5.4: Σύγκριση αποτελεσμάτων της παρούσας διπλωματικής (H2) και της μελέτης των Draganova et al (H1). Το δίκτυο ελέγχθηκε με θορυβώδεις εικόνες (περιπτώσεις 7-11)

	Cases 1-6		Cases 7-11
	Average of Overall Means	Average of Restricted Means	Average of Overall Means
H1	3.22	6.28	2.16
H2	**1.57**	6.95	**1.46**

Πίνακας 5.5: Σύγκριση των μέσων όρων των σφαλμάτων της παρούσας διπλωματικής (H2) και της μελέτης των Draganova et al (H1) για όλες τις περιπτώσεις (1-11)

Με βάση τις πιο πάνω μετρήσεις παρατηρούμε ότι τα αποτελέσματα των δύο μελετών για το δίκτυο Hopfield κυμαίνονται περίπου στις ίδιες τιμές. Σε κάποιες περιπτώσεις το σφάλμα της μελέτης που υλοποιήθηκε από τους Draganova et al είναι μικρότερο και κάποιες όχι. Ο λόγος που τα αποτελέσματα διαφέρουν, παρόλο που χρησιμοποιήθηκε η ίδια δομή δικτύου και τα ίδια δεδομένα εισόδου, είναι τα διαφορετικά επικαλυμμένα και θορυβώδη σημεία που χρησιμοποιήθηκαν στον έλεγχο του δικτύου. Στην μελέτη [4] χρησιμοποιήθηκαν συγκεκριμένα σημεία, ενώ στη παρούσα διπλωματική τα σημεία που θα επικαλύπτονταν επιλέγονταν είτε τυχαία (περιπτώσεις θορύβου 7-11) είτε με τον τρόπο που ακολουθεί (περιπτώσεις 1-6): Η κάθε εικόνα προσώπου χωριζόταν σε τέσσερα τεταρτημόρια με βάση το μέσο της μέγιστης συντεταγμένης x και y και αναλόγως της κάθε περίπτωσης επικαλυπτόταν και το αντίστοιχο τεταρτημόριο. Στο σχήμα 5.5 φαίνεται ένα παράδειγμα εικόνας προσώπου που είναι χωρισμένη σε τέσσερα τεταρτημόρια.



Σχήμα 5.5: Κάθε εικόνα προσώπου χωρίζεται σε τέσσερα τεταρτημόρια και αναλόγως της περίπτωσης επικαλύπτεται το αντίστοιχο τεταρτημόριο

Ο θόρυβος, στις περιπτώσεις 7-11, προστίθεται με την αντικατάσταση τυχαίων σημείων με τυχαίες τιμές που βρίσκονται στο διάστημα μεταξύ των μέγιστων και ελάχιστων συντεταγμένων x και y, της συγκεκριμένης εικόνας. Η επικάλυψη σημείων επιτυγχάνεται με την αντικατάσταση των σημείων, που βρίσκονται στο τεταρτημόριο που πρέπει να επικαλυφθεί, επίσης με τυχαίες τιμές που βρίσκονται στο διάστημα μεταξύ των μέγιστων και ελάχιστων συντεταγμένων x και y, της συγκεκριμένης εικόνας. Άρα, εφόσον οι εικόνες προσώπου που χρησιμοποιούνται για τον έλεγχο του δικτύου έχουν διαφορετικά σημεία επικαλυμμένα ή με θόρυβο και με διαφορετικές τιμές, δεν μπορούν τα αποτελέσματα των δύο μελετών να είναι άμεσα συγκρίσιμα.

Κεφάλαιο 6

Συμπεράσματα

6.1 Σύνοψη

Η αποκατάσταση μερικώς επικαλυμμένων εικόνων προσώπου με νευρωνικά δίκτυα είναι ένα θέμα που παρουσιάζει αρκετό ενδιαφέρον στον τομέα της επιστήμης της πληροφορικής. Αρκετές μελέτες έχουν υλοποιηθεί που επιλύουν το συγκεκριμένο πρόβλημα με διάφορα είδη νευρωνικών δικτύων.

Στην παρούσα διπλωματική έχει γίνει προσπάθεια για την επίλυση του προβλήματος αποκατάστασης μερικώς επικαλυμμένων εικόνων προσώπου με νευρωνικά δίκτυα Hopfield και μηχανές Boltzmann. Ο λόγος που επιλέχθηκαν τα δύο αυτά συγκεκριμένα νευρωνικά δίκτυα είναι γιατί έχουν την ιδιότητα να συσχετίζουν πρότυπα και αυτό ακριβώς χρειάζεται η συγκεκριμένη εφαρμογή για να επιλυθεί. Τα δύο δίκτυα, αφού εκπαιδεύτηκαν με τα δεδομένα εκπαίδευσης, ήταν ικανά, δεδομένης μιας ελλιπής ή θορυβώδης εικόνας προσώπου, να συσχετίζουν την εικόνα αυτή με την κοντινότερη, από αυτές που είναι ήδη αποθηκευμένες στο δίκτυο, ενεργειακά εικόνα προσώπου και να την ανακτούν.

Τα αποτελέσματα της υλοποίησης της εφαρμογής με το δίκτυο Hopfield ήταν σχετικά ικανοποιητικά. Σύμφωνα όμως και με τη θεωρία, τα δίκτυα Hopfield αντιμετωπίζουν προβλήματα χωρητικότητας και το πρόβλημα των τοπικών ελαχίστων. Οπόταν είναι φυσικό και αναμενόμενο να μην έχουν τόσο καλά αποτελέσματα όσο άλλα νευρωνικά δίκτυα. Τα δίκτυα Boltzmann, αντιθέτως, σύμφωνα με τη θεωρία πρέπει να έχουν καλύτερα αποτελέσματα από τα δίκτυα Hopfield, εφόσον αντιμετωπίζουν τα προβλήματα χωρητικότητας και τοπικών ελαχίστων. Όμως, λόγω έλλειψης χρόνου δεν καταφέραμε να το αποδείξουμε στην παρούσα διπλωματική.

Αν και μη ολοκληρωμένη όπως πρέπει η παρούσα διπλωματική, εντούτοις μπορούμε να διαπιστώσουμε και να συμπεράνουμε ότι το πρόβλημα της αποκατάστασης μερικώς επικαλυμμένων εικόνων μπορεί να λυθεί αποδοτικά με νευρωνικά δίκτυα και συγκεκριμένα με δίκτυα Hopfield και μηχανές Boltzmann. Έχουμε αποδείξει ότι ακόμα και αν ένα μεγάλο μέρος της εικόνας προσώπου είναι επικαλυμμένο ή θορυβώδης, το δίκτυο είναι ικανό να μας ανακτήσει μια αρκετά κοντινή ενεργειακά εικόνα προσώπου.

6.2 Μελλοντική Εργασία

Λόγω της καθυστέρησης ολοκλήρωσης της υλοποίησης του δικτύου Hopfield αλλά και λόγω του ότι το δίκτυο Boltzmann είναι εξαιρετικά αργό, το πρόβλημα με το δίκτυο Boltzmann δεν έχει λυθεί όπως ήταν αρχικά προγραμματισμένο. Το δίκτυο Boltzmann έχει υλοποιηθεί, όμως τα αποτελέσματα δεν είναι ικανοποιητικά. Στο μέλλον θα μπορούσαν να γίνουν κάποιες αλλαγές, ούτως ώστε να επιτευχθούν καλύτερα αποτελέσματα. Καταρχάς, θα πρέπει να αυξηθούν οι επαναλήψεις εκπαίδευσης του δικτύου και ακολούθως να εξεταστεί το κατά πόσο πρέπει να αλλάζουν τα βάρη κάθε φορά. Επίσης, θα πρέπει να ελεγχθούν τα αποτελέσματα του δικτύου για τις υπόλοιπες περιπτώσεις που φαίνονται στα σχήματα 2.3 και 2.4.

Βιβλιογραφία

[1] David H. Ackley, Geoffrey E. Hinton and Terrence J. Sejnowski, "A Learning Algorithm for Boltzmann Machines", Cognitive Science, vol. 9, Issue 1, pp. 147-169, 1985.

[2] Beale, R. and Jackson, T., *Neural Computing: An Introduction*, Bristol and Philadelphia: Institute of Physics Publishing, 1990

[3] Kim T. Blackwell, Thomas P. Vogl, Hans P. Dettmar, Michael A. Brown, Garth S. Barbour, Daniel L. Alkon, "Identification of faces obscured by noise: comparison of an artificial neural network with human observers", J. Expt. Theor. Artif. Intell., pp. 491-508, 1997.

[4] D. Draganova, A. Lanitis and C. Christodoulou, "Restoration of Partially Occluded Shapes of Faces using Neural Networks", Computer Recognition Systems, Advances in Soft Computing , ed. by M. Kurzynski, E. Puchala, M. Wozniak, A. Zolnierek, Springer, pp. 767-774, 2005.

[5] Scott E. Fahlman, Geoffrey E. Hinton and Terrence J. Sejnowski, "Massively Parallel Architectures for AI: NETL, Thistle, and Boltzmann Machines", Proceedings of the AAAI-83 conference, 1983.

[6] K. Fukushima, "Restoring partly occluded patterns: a neural network model", Neural Networks, 18(1), in press, 2005.

[7] Geoffrey E. Hinton and Terrence J. Sejnowski, "Optimal Perceptual Inference", Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1983.

[8] J.J. Hopfield, “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”, Proc. Nat. Acad. Sci. USA, vol 79, pp. 2554-2558, 1982.

[9] W. Zhao, P.J. Phillips, R. Chellappa, A. Rosenfeld, “Face Recognition: A Literature Survey”, ACM Computing Surveys, Vol. 35, No. 4, pp. 399-458, 2003.

[10] FG – NET Aging Database http://www.fgnet.rsunit.com/, 2004

Παράρτημα Α

Απόδειξη Υποκεφαλαίου 3.2

Η απόδειξη ότι όσο το δίκτυο πλησιάζει στη θερμική ισορροπία, οι χαμηλές ενεργειακές καταστάσεις είναι πιο πιθανές, είναι η εξής:

Σε κατάσταση θερμικής ισορροπίας, η πιθανότητα το δίκτυο να βρίσκεται στην κατάσταση a και b είναι, σύμφωνα με το Boltzmann Distribution, $P_a = k\, e^{-E_a / T}$ και $P_b = k\, e^{-E_b / T}$. Άρα, όταν το δίκτυο θα φτάσει σε θερμική ισορροπία η σχετική πιθανότητα των δύο καταστάσεων a και b θα είναι, σύμφωνα με το Boltzmann Distribution, $\frac{P_a}{P_b} = e^{-(E_a - E_b)/T}$, όπου P_a η πιθανότητα να βρίσκεται το δίκτυο στην κατάσταση a και P_b η πιθανότητα να βρίσκεται το δίκτυο στην κατάσταση b. Έτσι, αν

$$E_a < E_b$$

$$\Rightarrow e^{-(E_a - E_b)/T} > 1$$

$$\Rightarrow \frac{P_a}{P_b} > 1$$

$$\Rightarrow P_a > P_b$$

Παράρτημα Β

Κώδικας Δικτύου Hopfield

```
/* Filename: Hopfield.cpp
   Apostolou Lena */

#include <cstdlib>
#include <iostream>
#include <fstream>
#include <vector>
#include <math.h>
#include<cmath>
#include <string.h>
#include<ctime> //time function

using namespace std;

#define NUM_NEURONS 680
#define TRAIN_PAT 102
#define TEST_PAT 900

//Global variables
int weights[NUM_NEURONS][NUM_NEURONS];
int training_data[TRAIN_PAT][NUM_NEURONS];
int testing_data[TEST_PAT][NUM_NEURONS];
int noisy_data[TEST_PAT][NUM_NEURONS];
```

```
ofstream eout("Error.txt");
ofstream rout("Recovered Points.txt");
ofstream tout("testing.txt");
ofstream iopout("Initial Occluded Points.txt");
ofstream dout("Standard Deviation.txt");

//****************************** FUNCTIONS ****************************************//

//Creates occlusion on face image depending on the case
void CreateOcclusion(int table[68][2],int occ_case,int pat){

	int i;
	int minX,minY,maxX,maxY;
	int rangeX,rangeY;
	double medX,medY;
	int matrix[68][3];

	//Find minimum and maximum coordinates
	minX = table[0][0];
	maxX = table[0][0];
	minY = table[0][1];
	maxY = table[0][1];

	for (i=0;i<68;i++){
		if (table[i][0]<minX)
			minX = table[i][0];
		if (table[i][1]<minY)
			minY = table[i][1];
		if (table[i][0] > maxX)
			maxX = table[i][0];
		if (table[i][1] > maxY)
			maxY = table[i][1];
	}

	medX = (minX + maxX)/2;
	medY = (minY + maxY)/2;
	rangeX = (maxX - minX)+1;
	rangeY = (maxY - minY)+1;
```

```
switch (occ_case) {

case 1:
      for (i=0;i<68;i++){
            if ((table[i][0]<medX) && (table[i][1]>medY)){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 2:
      for (i=0;i<68;i++){
            if (table[i][0]<medX){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 3:
      for (i=0;i<68;i++){
```

```
            if (table[i][1]>medY){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;
case 4:
      for (i=0;i<68;i++){
            if ((table[i][0]>medX) || (table[i][1]<medY)){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 5:
      for (i=0;i<68;i++){
            if (table[i][0]>medX){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
```

```
                        matrix[i][2] = table[i][1];
                  }
                  else {      // not occluded points
                        matrix[i][0] = -1;
                        matrix[i][1] = -1;
                        matrix[i][2] = -1;
                  }
            }
      break;

      case 6:
            for (i=0;i<68;i++){
                  if (table[i][1]<medY){
                        table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                        table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                        matrix[i][0] = 1; // it is occluded point
                        matrix[i][1] = table[i][0];
                        matrix[i][2] = table[i][1];
                  }
                  else {      // not occluded points
                        matrix[i][0] = -1;
                        matrix[i][1] = -1;
                        matrix[i][2] = -1;
                  }
            }
      break;
      }

      for (int w=0;w<68;w++)
            iopout<<matrix[w][0]<<"\t"<<matrix[w][1]<<"\t"<<matrix[w][2]<<endl;
      iopout<<endl;
}
```

```
//Adds noise on face image depending on the case (noise)
void SetNoise(int tableXY[68][2],int noise,int pat){

      int i=0;
      int j=0;
      int rand_pos=0;
      int previous=0;
      int min,max;
      int range;
      vector<int> temp;
      vector<int> noise_data;

      //find minimum and maximum coordinate
      min = tableXY[0][0];
      max = tableXY[0][0];
      for (i=0;i<68;i++){
            if (tableXY[i][0] < min)
                  min = tableXY[i][0];
            else if (tableXY[i][1] < min)
                  min = tableXY[i][1];
            else if (tableXY[i][0] > max)
                  max = tableXY[i][0];
            else if (tableXY[i][1] > max)
                  max = tableXY[i][1];
      }

      j = (int)((noise * 68.0)/100.0);    //calculate how many points will be noisy
    range = (max - min)+1;
      rand_pos = (rand()%68); // finds the position of the table that will be changed

      for (i=0;i<j;i++){
            previous = rand_pos;
            rand_pos = (rand()%68);
            while (rand_pos==previous)
                  rand_pos = (rand()%68);

            tableXY[rand_pos][0] = min + int(range*rand()/(RAND_MAX + 1.0)); // add noise to coordinate x
```

```
            tableXY[rand_pos][1] = min + int(range*rand()/(RAND_MAX + 1.0)); // add noise to coordinate y

      }
}

// Converts decimal to binary
void dec2bin(long decimal, char *binary)
{
      int k = 0, n = 0;
      int neg_flag = 0;
      int remain;
      int old_decimal;
      char temp[100];

      // take care of negative input
      if (decimal < 0){
            decimal = -decimal;
            neg_flag = 1;
      }

      do{
            old_decimal = decimal;
            remain = decimal % 2;
            // whittle down the decimal number
            decimal = decimal / 2;
            // converts digit 0 or 1 to character '0' or '1'
            temp[k++] = remain + '0';

      } while (decimal > 0);

      if (neg_flag)
            temp[k++] = '-'; // add - sign
      else
            temp[k++] = ' '; // space
      while (k >= 0)
            binary[n++] = temp[--k];
      binary[n-1] = 0; // end with NULL
}
```

```
// Converts decimal points to binary vector
void NormalizePoints(int tableXY[68][2],int training,int pos,int add_noise){

	int i,temp,h,pointer;
	long x,y;
	int xi,yi;
	char binary[100];
	char Xfinal[6];
	char Yfinal[6];
	vector<int> tempV;

	for (i=0;i<68;i++){
		xi = tableXY[i][0];
		x = (xi + 107)/(214.0/31.0);  // scaling from [-107,107] to [0,31]
		yi = tableXY[i][1];
		y = (yi + 107)/(214.0/31.0);

		dec2bin(x,binary);	// convert decimal to binary
		temp = 0;
		while(binary[temp]!='\0'){
				temp++;
		}
		// check if binary's size is different than 5
		if ((temp-1)!=5){
			h = 0;
			while(h < (5 - (temp-1))){
				Xfinal[h] = '0';
				h++;
			}
			pointer = 1;
			while(h<5){
				Xfinal[h] = binary[pointer];
				h++;
				pointer++;
			}
		}
		else {
			for (h=0;h<5;h++)
```

```
                  Xfinal[h] = binary[h+1];
      }
      Xfinal[5] = 0;

      temp = 0;
      dec2bin(y,binary);      // convert to binary
      while(binary[temp]!='\0'){
            temp++;
      }
      // check if binary's size is different than 5
      if ((temp-1)!=5){
            h = 0;
            while(h < (5 - (temp-1))){
                  Yfinal[h] = '0';
                  h++;
            }
            pointer = 1;
            while(h<5){
                  Yfinal[h] = binary[pointer];
                  h++;
                  pointer++;
            }
      }
      else {
            for (h=0;h<5;h++)
                  Yfinal[h] = binary[h+1];
      }
      Yfinal[5] = 0;

      for(int f=0;f<5;f++)
            tempV.push_back((Xfinal[f])-48);    // convert from char to int
      for(int f=0;f<5;f++)
            tempV.push_back(Yfinal[f]-48);            // convert from char to int
}

if (training==1 && add_noise==0){   // its training
      for (int f=0;f<tempV.size();f++){
            if (tempV[f]==0)
```

```
                        training_data[pos][f] = -1;
                  else
                        training_data[pos][f] = 1;
            }
      }
      else if (training==0 && add_noise==0){    // its testing
            for (int f=0;f<tempV.size();f++){
                  if (tempV[f]==0)
                        testing_data[pos][f] = -1;
                  else
                        testing_data[pos][f] = 1;
            }
      }
      else if (training==0 && add_noise==1){
            for (int f=0;f<tempV.size();f++){
                  if (tempV[f]==0)
                        noisy_data[pos][f] = -1;
                  else
                        noisy_data[pos][f] = 1;
            }
      }
}

// Initializes Weights
void InitializeWeights(){

      int i,j,sum,w;

      for (i=0;i<NUM_NEURONS;i++){
            for (j=0;j<NUM_NEURONS;j++){
                  if (i!=j){
                        sum = 0;
                        for (w=0;w<TRAIN_PAT;w++)
                              sum = sum + (training_data[w][i]*training_data[w][j]); // Wij = Sum(Xi*Xj)
                        weights[i][j] = sum;
                  }
                  else
                        weights[i][j] = 0;
```

```
            }
      }
}

// Calculates hamming distance between noisy testing pattern and training pattern
int CalculateHammingDistance(int outputs[680],int original[680]){

 int dist = 0;
 for (int i = 0; i<680; i++){
          if ((outputs[i]!=original[i]))
            dist++;
 }
 return dist;
}

// Converts binary vector to decimal points
void CreatePoints(int output[680],int pattern){

      int x[6];
      int y[6];
      int sumX=0;
      int sumY=0;
      int i,j,k,l;
      int pointX,pointY;

      rout<<"Pattern "<<pattern<<endl<<endl;
      for (l=0;l<680;l=l+10){
            j=l;
            while(j<l+5){
                  x[j%5] = output[j];
                  j++;
            }
            x[5] = 0;

            while(j<l+10){
                  y[j%5] = output[j];
                  j++;
            }
```

```
            y[5] = 0;
            // until now calculates the x and y in bipolar form

            // convert x and y in binary form
            for (i=0;i<5;i++){
                  if (x[i]==-1)
                        x[i] = 0;
                  if (y[i]==-1)
                        y[i] = 0;
            }

            // converts from binary to decimal form
            k = 16;
            for (i=0;i<5;i++){
                  if (x[i]==1)
                        sumX = sumX + k;
                  k = k/2;
            }

            k = 16;
            for (i=0;i<5;i++){
                  if (y[i]==1)
                        sumY = sumY + k;
                  k = k/2;
            }

            // perform scaling from [0,31] to [-107,107]
            pointX = (int)(((double)sumX - 15.5) * (214.0/31.0));
            pointY = (int)(((double)sumY - 15.5) * (214.0/31.0));

            rout<<pointX<<"\t"<<pointY<<endl;

            sumX = 0;
            sumY = 0;
      }
      rout<<endl<<endl;
}
```

```
// Returns an integer between [low, high]
int random_int(int low,int high){

	int x = (int)((rand()%(high-low)) + low);
	return x;
}

// Test the network for convergence
void ConvergenceFunc(int outputs[680],int pattern,int original[680]){

	vector<int> temp_outputs;
	vector<int> print;
	int convergence = 0;
	int flag=0;
	double result=0.0;
	double thres = 0.0;
	int iter=0;
	int unit;
	int max_iter = 3000;
	int hamming;
	int net_sum;

	hamming = CalculateHammingDistance(outputs,original);

	while(iter<max_iter && hamming>0){

		iter++;
		unit = random_int(0,680);
		net_sum = 0;

		for (int j=0;j<680 ;j++ )
			net_sum = net_sum + weights[unit][j]*outputs[j];

		if (net_sum>0)
			outputs[unit] = 1;
		else if (net_sum<0)
			outputs[unit] = -1;
```

```
            hamming = CalculateHammingDistance(outputs,original);
      }
      CreatePoints(outputs,pattern);

      return;
}

/* Finds the training pattern with the smallest hamming distance
to the corresponding testing pattern */
int FindMinHamming(int noisy_vector[680]){

      int i,pos;
      int min_ham;
      int hamming;

      min_ham = CalculateHammingDistance(noisy_vector,training_data[0]);
      pos = 0;

      for(i=0;i<TRAIN_PAT;i++){
            hamming = CalculateHammingDistance(noisy_vector,training_data[i]);
            if (hamming<min_ham){
                  min_ham = hamming;
                  pos = i;
            }
      }
      return pos;
}
```

```
/*********************************************/
/****************** MAIN **********************/
/*********************************************/
int _tmain(int argc, _TCHAR* argv[])
{
      int snum=0;
      char sample[15];
      long xi,yi;
      int h = 0;
      int i=0;
      int in=0;
      int st=0;
      int temp=0;
      char x[15];
      char y[15];
      int table [68][2];

      srand((unsigned)time(NULL));

      FILE *fin;
      fin = fopen("pam102.PTS","r");
      if (fin==NULL){
            cout <<"This file doesn't exist"<<"\n";
            exit(-1);
      }

      while (!feof(fin)){
            fscanf(fin,"%s",sample);      // read the word Sample
            fscanf(fin,"%d",&snum);       // read the number next to Sample
            for(i=0;i<68;i++){
                  fscanf(fin,"%s",x);     // read the first coordinate as string
                  xi = atoi(x);           // convert string to integer
                  // Same for coordinate y
                  fscanf(fin,"%s",y);
                  yi = atoi(y);
                  // insert the coordinates(x,y) in table
                  table[i][0] = xi;
                  table[i][1] = yi;
```

```
        }
        NormalizePoints(table,1,snum-1,0);  // 1 means its for training(use the training_data matrix)
  }
  fclose(fin);

  //read the case (1-11) from file
  FILE *finCases;
  finCases = fopen("Case.txt","r");
  if (finCases== NULL){
   cout <<"This file doesn't exist"<<"\n";
   exit(-1);
}

  int cas,noise;
  fscanf(finCases,"%d",&cas);
  if (cas>=7 && cas<=11)        // For cases 7-11 we need to know the noise too
        fscanf(finCases,"%d",&noise);
  fclose(finCases);

  //Read testing data
FILE *fin900;
  fin900 = fopen("pam900.PTS","r");
  if (fin900== NULL){
   cout <<"This file doesn't exist"<<"\n";
   exit(-1);
}

  while ((!feof(fin900))){
        fscanf(fin900,"%s",sample);   // read the word Sample
        fscanf(fin900,"%d",&snum);    // read the number next to Sample
        for(i=0;i<68;i++){
              fscanf(fin900,"%s",x);
              xi = atoi(x);
              fscanf(fin900,"%s",y);
              yi = atoi(y);
              table[i][0] = xi;
              table[i][1] = yi;
        }
```

```
      NormalizePoints(table,0,snum-103,0);      // 0 its for testing(use the testing_data matrix)
                                                // 0 its for not adding noise
      if (cas>=7 && cas<=11)
            SetNoise(table,noise,snum-103);     // function that adds noise on faces
      else if (cas>=1 && cas<=6)
            CreateOcclusion(table,cas,snum-103);      // function that creates occlusion on faces

      NormalizePoints(table,0,snum-103,1);      // 0 its using the noisy_data matrix)
                                                // 1 its for adding noise
}

/****************** Initialize Weights *********************/
InitializeWeights();


/*********************** Testing *************************/
int pos;
for (int j=0;j<TEST_PAT;j++){
      pos = FindMinHamming(noisy_data[j]);
      ConvergenceFunc(noisy_data[j],j,training_data[pos]);
}

/********************** CALCULATE ERROR ********************/
FILE *err1;
err1 = fopen("Initial Points.txt","r");
if (err1==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

FILE *err2;
err2 = fopen("Recovered Points.txt","r");
if (err2==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

FILE *err3;
```

```
err3 = fopen("Initial Points.txt","r");
if (err3==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

FILE *err4;
err4 = fopen("Recovered Points.txt","r");
if (err4==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

FILE *restr = fopen("Initial Occluded Points.txt","r");
if (restr==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

FILE *restr2 = fopen("Initial Occluded Points.txt","r");
if (restr2==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}

char pattern[15];
int pnum;
int pat;
int count_pat;
int x1,y1,x2,y2;
int x3,y3;
double error,rerror,o_deviation,r_deviation;
double total_error,total_rerror;
double od_mean,rd_mean;
vector<double> r_di,o_di;
int if_occluded;
int count_occ =0;
double o_sum,r_sum;
```

```
total_error = 0;
total_rerror = 0;
count_pat = 0;
o_sum = 0.0;
r_sum = 0.0;
o_deviation =0;
r_deviation = 0;

while ( (!feof(err1)) && count_pat<900){

      count_pat++;
      fscanf(err1,"%s",pattern);    // read the word Pattern
      fscanf(err2,"%s",pattern);    // read the word Pattern
      fscanf(err1,"%d",&pnum);      // read the number next to Pattern
      fscanf(err2,"%d",&pnum);      // read the number next to Pattern
      rerror = 0;
      error = 0;
      count_occ =0;

      for(pat=0;pat<68;pat++){
            // read from "Initial Points.txt" x1 and y1
            fscanf(err1,"%d",&x1);
            fscanf(err1,"%d",&y1);
            // read from "Recovered Points.txt" x2 and y2
            fscanf(err2,"%d",&x2);
            fscanf(err2,"%d",&y2);

            if (cas>=1 && cas<=6){
                  // read from "Initial Occluded Points.txt" x3 and y3
                  fscanf(restr,"%d",&if_occluded);
                  fscanf(restr,"%d",&x3);
                  fscanf(restr,"%d",&y3);
                  if (if_occluded==1){
                        count_occ++;
                        rerror = rerror + pow(double(x2-x3),2) + pow(double(y2-y3),2);
                  }
            }
            error = error + (pow(double(x2-x1),2) + pow(double(y2-y1),2));
```

```
	}

	error = (double)sqrt((double)error);
	error = (double)error/(double)68.0;
	total_error = total_error + error; //total overall error

	if (cas>=1 && cas<=6){
		rerror = (double)sqrt((double)rerror);
		rerror = (double)rerror/(double)68.0;
		total_rerror = total_rerror + rerror; //total restricted error
	}
}
fclose(err1);
fclose(err2);
fclose(restr);

total_error = (double)total_error/(double)count_pat;
eout<<"Mean Overall Error = "<<total_error<<endl;
count_pat = 0;

count_occ=0;
while ( (!feof(err3)) && count_pat<900 ){

	count_pat++;
	fscanf(err3,"%s",pattern);    // read the word Pattern
	fscanf(err4,"%s",pattern);    // read the word Pattern
	fscanf(err3,"%d",&pnum);      // read the number next to Pattern
	fscanf(err4,"%d",&pnum);      // read the number next to Pattern
	// calculate standard deviation of overall error
	for(pat=0;pat<68;pat++){
		// read from "Initial Points.txt" x1 and y1
		fscanf(err3,"%d",&x1);
		fscanf(err3,"%d",&y1);
		// read from "Recovered Points.txt" x2 and y2
		fscanf(err4,"%d",&x2);
		fscanf(err4,"%d",&y2);
		o_sum = o_sum + sqrt((double)(pow(double(x2-x1),2) + pow(double(y2-y1),2)));
```

```
            if (cas>=1 && cas<=6){
                  fscanf(restr2,"%d",&if_occluded);
                  fscanf(restr2,"%d",&x3);
                  fscanf(restr2,"%d",&y3);
                  if (if_occluded==1){
                        count_occ++;
                        r_sum = r_sum + sqrt(pow(double(x2-x3),2) + pow(double(y2-y3),2));

                  }
            }
      }
      o_sum = (double)o_sum/(double)68.0;
      o_di.push_back(o_sum);

      if (cas>=1 && cas<=6){
            r_sum = (double)r_sum/(double)68.0;
            r_di.push_back(r_sum);
      }
      o_sum = 0;
      r_sum = 0;
      count_occ = 0;
}
od_mean = 0.0;
rd_mean = 0.0;

for (int f=0;f<o_di.size();f++){
      od_mean = od_mean + o_di[f];
      if (cas>=1 && cas<=6)
            rd_mean = rd_mean + r_di[f];
}
od_mean = od_mean / (double)900.0;
if (cas>=1 && cas<=6)
      rd_mean = rd_mean / (double)900.0;

for (int f=0;f<o_di.size();f++){
      o_deviation = o_deviation + (o_di[f] - od_mean)* (o_di[f] - od_mean);
      if (cas>=1 && cas<=6)
            r_deviation = r_deviation + (r_di[f] - rd_mean)* (r_di[f] - rd_mean);
```

```
	}
	o_deviation = o_deviation/(double)900.0;
	o_deviation = sqrt(o_deviation);

	if (cas>=1 && cas<=6){
		r_deviation = r_deviation/(double)900.0;
		r_deviation = sqrt(r_deviation);
	}
	dout<<"Standard Deviation of Overall Error = "<<o_deviation<<endl;

	if (cas>=1 && cas<=6){
		total_rerror = (double)total_rerror/(double)count_pat;
		eout<<"Mean Restricted Error = "<<total_rerror<<endl;
		dout<<"Standard Deviation of Restricted Error = "<<r_deviation<<endl;
	}

	fclose(err3);
	fclose(err4);
	fclose(restr2);
	fclose(fin900);
	cout<<endl<<"Exiting..."<<endl;
	return 0;
}
```

Παράρτημα Γ

Κώδικας Δικτύου Boltzmann

```
/* Filename: Boltzmann.cpp
   Apostolou Lena */

#include <cstdlib>
#include <iostream>
#include <fstream>
#include <vector>
#include <math.h>
#include<cmath>
#include <string.h>
#include<ctime>

using namespace std;

#define inp_units       680
#define out_units       680
#define hidden_units    700
#define units           2060  // inp_units + out_units + hidden_units
#define TRAIN_PAT 102
#define TEST_PAT 900

//Global variables
int training_data[TRAIN_PAT + TRAIN_PAT][680];
int testing_data[TEST_PAT][680];
int noisy_data[TEST_PAT][680];
```

```
double weights[units][units];
int changes=0;

ofstream tout("testing.txt");
ofstream dout("Standard Deviation.txt");
ofstream iopout("Initial Occluded Points.txt");
ofstream eout("Error.txt");

//****************************** FUNCTIONS ****************************************//

//Creates occlusion on face image depending on the case
void CreateOcclusion(int table[68][2],int occ_case){

      int i;
      int minX,minY,maxX,maxY;
      int rangeX,rangeY;
      double medX,medY;
      int matrix[68][3];

      minX = table[0][0];
      maxX = table[0][0];
      minY = table[0][1];
      maxY = table[0][1];
      for (i=0;i<68;i++){
            if (table[i][0]<minX)
                  minX = table[i][0];
            if (table[i][1]<minY)
                  minY = table[i][1];
            if (table[i][0] > maxX)
                  maxX = table[i][0];
            if (table[i][1] > maxY)
                  maxY = table[i][1];
      }
      medX = (minX + maxX)/2;
      medY = (minY + maxY)/2;
      rangeX = (maxX - minX)+1;
      rangeY = (maxY - minY)+1;
```

```
switch (occ_case) {

case 1:
      for (i=0;i<68;i++){
            if ((table[i][0]<medX) && (table[i][1]>medY)){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 2:
      for (i=0;i<68;i++){
            if (table[i][0]<medX){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 3:
      for (i=0;i<68;i++){
```

```
            if (table[i][1]>medY){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 4:
      for (i=0;i<68;i++){
            if ((table[i][0]>medX) || (table[i][1]<medY)){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
                  matrix[i][1] = table[i][0];
                  matrix[i][2] = table[i][1];
            }
            else {      // not occluded points
                  matrix[i][0] = -1;
                  matrix[i][1] = -1;
                  matrix[i][2] = -1;
            }
      }
break;

case 5:
      for (i=0;i<68;i++){
            if (table[i][0]>medX){
                  table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                  table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                  matrix[i][0] = 1; // it is occluded point
```

```
                        matrix[i][1] = table[i][0];
                        matrix[i][2] = table[i][1];
                  }
                  else {      // not occluded points
                        matrix[i][0] = -1;
                        matrix[i][1] = -1;
                        matrix[i][2] = -1;
                  }
            }
      break;

      case 6:
            for (i=0;i<68;i++){
                  if (table[i][1]<medY){
                        table[i][0] = minX + int(rangeX*rand()/(RAND_MAX + 1.0));
                        table[i][1] = minY + int(rangeY*rand()/(RAND_MAX + 1.0));
                        matrix[i][0] = 1; // it is occluded point
                        matrix[i][1] = table[i][0];
                        matrix[i][2] = table[i][1];
                  }
                  else {      // not occluded points
                        matrix[i][0] = -1;
                        matrix[i][1] = -1;
                        matrix[i][2] = -1;
                  }
            }
      break;
      }

      for (int w=0;w<68;w++)
            iopout<<matrix[w][0]<<"\t"<<matrix[w][1]<<"\t"<<matrix[w][2]<<endl;
      iopout<<endl;
}
```

```
//Adds noise on face image depending on the case (noise)
void SetNoise(int tableXY[68][2],int noise){

      int i=0;
      int j=0;
      int rand_pos=0;
      int previous=0;
      int min,max;
      int range;
      vector<int> temp;
      vector<int> noise_data;

      //find minimum and maximum coordinate
      min = tableXY[0][0];
      max = tableXY[0][0];
      for (i=0;i<68;i++){
            if (tableXY[i][0] < min)
                  min = tableXY[i][0];
            else if (tableXY[i][1] < min)
                  min = tableXY[i][1];
            else if (tableXY[i][0] > max)
                  max = tableXY[i][0];
            else if (tableXY[i][1] > max)
                  max = tableXY[i][1];
      }

      j = (int)((noise * 68.0)/100.0);    //calculate how many points will be noisy
    range = (max - min)+1;
      rand_pos = (rand()%68);
      for (i=0;i<j;i++){
            previous = rand_pos;
            rand_pos = (rand()%68);
            while (rand_pos==previous)
                  rand_pos = (rand()%68);
            tableXY[rand_pos][0] = min + int(range*rand()/(RAND_MAX + 1.0));  // add noise to coordinate x
            tableXY[rand_pos][1] = min + int(range*rand()/(RAND_MAX + 1.0));  // add noise to coordinate y
      }
}
```

```c
// Converts decimal to binary
void dec2bin(long decimal, char *binary)
{
	int k = 0, n = 0;
	int neg_flag = 0;
	int remain;
	int old_decimal;
	char temp[100];


	// take care of negative input
	if (decimal < 0){
		decimal = -decimal;
		neg_flag = 1;
	}
	do{
		old_decimal = decimal;
		remain = decimal % 2;
		// whittle down the decimal number
		decimal = decimal / 2;
		// converts digit 0 or 1 to character '0' or '1'
		temp[k++] = remain + '0';
	} while (decimal > 0);

	if (neg_flag)
		temp[k++] = '-'; // add - sign
	else
		temp[k++] = ' '; // space
	while (k >= 0)
		binary[n++] = temp[--k];
	binary[n-1] = 0; // end with NULL
}
```

```
// Converts decimal points to binary vector
void NormalizePoints(int tableXY[68][2],int training,int pos,int add_noise){

	int i,temp,h,pointer;
	long x,y;
	int xi,yi;
	char binary[100];
	char Xfinal[6];
	char Yfinal[6];
	vector<int> tempV;

	for (i=0;i<68;i++){
		xi = tableXY[i][0];
		x = (xi + 107)/(214.0/31.0);  // scaling from [-107,107] to [0,31]
		yi = tableXY[i][1];
		y = (yi + 107)/(214.0/31.0);

		dec2bin(x,binary);	// convert decimal to binary
		temp = 0;
		while(binary[temp]!='\0'){
				temp++;
		}
		// check if binary's size is different than 5
		if ((temp-1)!=5){
			h = 0;
			while(h < (5 - (temp-1))){
				Xfinal[h] = '0';
				h++;
			}
			pointer = 1;
			while(h<5){
				Xfinal[h] = binary[pointer];
				h++;
				pointer++;
			}
		}
		else {
			for (h=0;h<5;h++)
```

```
                  Xfinal[h] = binary[h+1];
      }
      Xfinal[5] = 0;

      temp = 0;
      dec2bin(y,binary);      // convert to binary
      while(binary[temp]!='\0'){
            temp++;
      }
      // check if binary's size is different than 5
      if ((temp-1)!=5){
            h = 0;
            while(h < (5 - (temp-1))){
                  Yfinal[h] = '0';
                  h++;
            }
            pointer = 1;
            while(h<5){
                  Yfinal[h] = binary[pointer];
                  h++;
                  pointer++;
            }
      }
      else {
            for (h=0;h<5;h++)
                  Yfinal[h] = binary[h+1];
      }
      Yfinal[5] = 0;

      for(int f=0;f<5;f++)
            tempV.push_back((Xfinal[f])-48);    // convert from char to int
      for(int f=0;f<5;f++)
            tempV.push_back(Yfinal[f]-48);            // convert from char to int
}

if (training==1 && add_noise==0){   // its training
      for (int f=0;f<tempV.size();f++){
            if (tempV[f]==0)
```

```
                        training_data[pos][f] = -1;
                  else
                        training_data[pos][f] = 1;
            }
      }
      else if (training==0 && add_noise==0){    // its testing
            for (int f=0;f<tempV.size();f++){
                  if (tempV[f]==0)
                        testing_data[pos][f] = -1;
                  else
                        testing_data[pos][f] = 1;
            }
      }
      else if (training==0 && add_noise==1){
            for (int f=0;f<tempV.size();f++){
                  if (tempV[f]==0)
                        noisy_data[pos][f] = -1;
                  else
                        noisy_data[pos][f] = 1;
            }
      }
}

// Initializes Weights
void InitializeWeights(){

      int min = -1;
      int max = 0;
      double x;
      int i = 0;
      int j = 0;

      ifstream win;

      win.open("weights.txt"); // opens the file
      if(!win) { // file couldn't be opened
            cerr << "Error: file could not be opened" << endl;
            exit(1);
```

```
      }

      double num;
      while ((!win.eof())&& (i<units)) { // keep reading until end-of-file
            for (j=0;j<units;j++){
                  win >> num;
                  weights[i][j] = num;
            }
            i++;
      }
      win.close();
}

// Returns a double number between [0,1]
double random_double(){
      return (rand()/(RAND_MAX + 1.0));
 }

// Initializes hidden neurons with random values
void InitializeHidden(int network[units],int pat){

      for (int i = inp_units + out_units;i < units;i++){
            if (random_double()>0.5)
                  network[i] = 1;
             else
                  network[i] = -1;
    }
}

// Clamps input and output neurons
void Clamp_inout(int network[units],int pat){

      int i;

      for (i=0;i < inp_units;i++)
            network[i] = training_data[pat][i]; // clamp input vector
      for (i=inp_units;i < inp_units + out_units;i++)
            network[i] = training_data[pat + TRAIN_PAT][i]; // clamp output vector
```

```
	InitializeHidden(network,pat);
}

// Returns an integer between [low, high]
int random_int(int low,int high){

	int x = (int)((rand()%(high-low)) + low);
	return x;
}

// Calculates output of neuron with probabilistic way
void CalculateNeuronOutput(int unit,int network[units],double T){

	double energy = 0.0;
	double probability;

	for (int i = 0; i < units;i++){
		if (i!=unit)
			energy = energy + weights[unit][i]*network[i];
	}

	probability = 1.0 / (double)(1 + exp(double(-energy)/(double)T));

	if (random_double()<=probability)
		network[unit] = 1;
	else
		network[unit] = -1;
}

// Stimulated Annealing for the positive phase
void StimAnnealPhase1(int network[units]){

	double Tinitial = 20;
	double Tfinal = 0.005;
	double a = 0.99;
	double T;

	T = Tinitial;
```

```
	for (int process_cycles=0;process_cycles<10;process_cycles++){
		for (int i = 0;i < hidden_units;i++){
			int unit = random_int(inp_units + out_units,units);
			CalculateNeuronOutput(unit,network,T);
		}
	}
	T = T*a;	// Temperature is reduced
	while (T > Tfinal){
		for (int process_cycles=0;process_cycles<10;process_cycles++){
			for (int i = 0;i < hidden_units;i++){
				int unit = random_int(inp_units + out_units,units);
				CalculateNeuronOutput(unit,network,T);
			}
		}
		T = T*a;
	}
}

// Weights between neurons that are on are updated
void UpdateWeights(int network[units],int phase){

	int output[680];
	double d = 0.005;

	for (int k =inp_units;k < inp_units + out_units;k++)
		output[k-inp_units] = network[k];   // assign in output the output vector

	for (int i = 0;i<units;i++){
		for (int j = 0;j<units;j++){
			if ((i!=j) && (output[i]==1) && (output[j]==1)){
				if (phase==1){
					weights[i][j] = weights[i][j] + d;
					changes++;
				}
				else if (phase==2){
					weights[i][j] = weights[i][j] - d;
					changes++;
				}
```

```
                  }
            }
      }
}

// Positive phase
void PositivePhase(int network[units]){

      for (int pat = 0;pat < TRAIN_PAT;pat++){
            Clamp_inout(network,pat);
            StimAnnealPhase1(network);
            UpdateWeights(network,1);
      }
}

// Initializes output and hidden neurons with random values
void InitializeOutputHidden(int network[units],int pat){

      for (int i = inp_units;i < units;i++){
            if (random_double()>0.5)
                  network[i] = 1;
             else
                  network[i] = -1;

      }
}

// Clamps input neurons
void Clamp_input(int network[units],int pat){

      for (int i=0;i < inp_units;i++)
            network[i] = training_data[pat][i]; // clamp input vector

      InitializeOutputHidden(network,pat);
}
```

```
// Stimulated Annealing for negative phase
void StimAnnealPhase2(int network[units]){

      double Tinitial = 20;
      double Tfinal = 0.005;
      double a = 0.99;
      double T;

      T = Tinitial;
      for (int process_cycles=0;process_cycles<10;process_cycles++){
            for (int i = 0;i < out_units + hidden_units;i++){
                  int unit = random_int(inp_units,units);
                  CalculateNeuronOutput(unit,network,T);
            }
      }
      T = T*a;
      while (T > Tfinal){
            for (int process_cycles=0;process_cycles<10;process_cycles++){
                  for (int i = 0;i < out_units + hidden_units;i++){
                        int unit = random_int(inp_units,units);
                        CalculateNeuronOutput(unit,network,T);
                  }
            }
            T = T*a;
      }
}

// Negative phase
void NegativePhase(int network[units]){

      for (int pat = 0;pat < TRAIN_PAT;pat++){
            Clamp_input(network,pat);
            StimAnnealPhase2(network);
            UpdateWeights(network,2);
      }
}
```

```
// Converts binary vector to decimal points
void CreatePoints(int output[680],int pattern,ofstream& rout){

	int x[6];
	int y[6];
	int sumX=0;
	int sumY=0;
	int i,j,k,l;
	int pointX,pointY;

	rout<<"Pattern "<<pattern<<endl<<endl;
	for (l=0;l<680;l=l+10){
		j=l;
		while(j<l+5){
			x[j%5] = output[j];
			j++;
		}
		x[5] = 0;

		while(j<l+10){
			y[j%5] = output[j];
			j++;
		}
		y[5] = 0;
		// until now calculates the x and y in bipolar form

		// convert x and y in binary form
		for (i=0;i<5;i++){
			if (x[i]==-1)
				x[i] = 0;
			if (y[i]==-1)
				y[i] = 0;
		}

		// convert from binary to decimal form
		k = 16;
		for (i=0;i<5;i++){
			if (x[i]==1)
```

```
                        sumX = sumX + k;
                  k = k/2;
            }

            k = 16;
            for (i=0;i<5;i++){
                  if (y[i]==1)
                        sumY = sumY + k;
                  k = k/2;
            }

            // perform scaling from [0,31] to [-107,107]
            pointX = (int)(((double)sumX - 15.5) * (214.0/31.0));
            pointY = (int)(((double)sumY - 15.5) * (214.0/31.0));

            rout<<pointX<<"\t"<<pointY<<endl;

            sumX = 0;
            sumY = 0;
      }
      rout<<endl<<endl;
}

// Calculates hamming distance between noisy testing pattern and testing pattern without noise
int CalculateHammingDistance(int outputs[680],int original[680]){
 int dist = 0;
 for (int i = 0; i<680; i++){
          if ((outputs[i]!=original[i]))
            dist++;
 }
 return dist;
}
```

```
// Test the network for convergence
void ConvergenceFunc(int outputs[680],int pattern,int original[680],ofstream& rout){

	vector<int> temp_outputs;
	vector<int> print;
	int convergence = 0;
	int flag=0;
	double result=0.0;
	double thres = 0.0;
	int iter=0;
	int unit;
	int max_iter = 3000;
	int hamming;
	double net_sum;

	hamming = CalculateHammingDistance(outputs,original);
	while(iter<max_iter && hamming>0){

		iter++;
		unit = random_int(0,680);
		net_sum = 0;

		for (int j=0;j<680 ;j++ )
			net_sum = net_sum + weights[unit][j]*outputs[j];

		if (net_sum>0)
			outputs[unit] = 1;
		else if (net_sum<0)
			outputs[unit] = -1;

		hamming = CalculateHammingDistance(outputs,original);
	}
	CreatePoints(outputs,pattern,rout);
	return;
}
```

```
/*********************************************/
/****************** MAIN **********************/
/*********************************************/
int main(int argc, char* argv[])
{
      int snum=0;
      char sample[15];
      long xi,yi;
      int h = 0;
      int in=0;
      int st=0;
      int temp=0;
      char x[15];
      char y[15];
      int table [68][2];

      srand((unsigned)time(NULL));

      FILE *fin;
      fin = fopen("pam102.pts","r");
      if (fin==NULL){
            cout <<"This file doesn't exist"<<"\n";
            exit(-1);
      }

      while (!feof(fin)){
            fscanf(fin,"%s",sample);      // read the word Sample
            fscanf(fin,"%d",&snum);       // read the number next to Sample
            for(int i=0;i<68;i++){
                  fscanf(fin,"%s",x);           // read the first coordinate as string
                  xi = atoi(x);                 // convert string to integer
                  // Same for coordinate y
                  fscanf(fin,"%s",y);
                  yi = atoi(y);
                  // insert the coordinates(x,y) in table
                  table[i][0] = xi;
                  table[i][1] = yi;
            }
```

```
        NormalizePoints(table,1,snum-1,0);  // 1 means its for training(use the training_data matrix)

  }
  fclose(fin);

  // until now training_data has only the input vectors
  // the output vectors will be exactly the same as input vectors
  for (int i = TRAIN_PAT;i < TRAIN_PAT + TRAIN_PAT;i++ ){
        for (int j = 0;j<680;j++)
              training_data[i][j] = training_data[i-TRAIN_PAT][j];
  }

  //read the case (1-11) from file
  int cas,noise;
  FILE *finCases;

  finCases = fopen("Case.txt","r");
  if (finCases== NULL){
   cout <<"This file doesn't exist"<<"\n";
   exit(-1);
}
  fscanf(finCases,"%d",&cas);
  if (cas>=7 && cas<=11)        // For cases 7-11 we need to know the noise too
        fscanf(finCases,"%d",&noise);
  fclose(finCases);


  //Read data for testing
FILE *fin900;
  fin900 = fopen("pam900.pts","r");
  if (fin900== NULL){
   cout <<"This file doesn't exist"<<"\n";
   exit(-1);
}

  int i;
  while ((!feof(fin900))){
        fscanf(fin900,"%s",sample);         // read the word Sample
```

```
        fscanf(fin900,"%d",&snum);          // read the number next to Sample
        for(i=0;i<68;i++){
              fscanf(fin900,"%s",x);
              xi = atoi(x);
              fscanf(fin900,"%s",y);
              yi = atoi(y);
              table[i][0] = xi;
              table[i][1] = yi;
        }
        NormalizePoints(table,0,snum-103,0);      // 0 its for testing(use the testing_data matrix)
                                                  // 0 its for not adding noise
        if (cas>=7 && cas<=11)
              SetNoise(table,noise);              //function that adds noise on faces
        else if (cas>=1 && cas<=6)
              CreateOcclusion(table,cas);         // function that creates occlusion on faces
        NormalizePoints(table,0,snum-103,1);      // 0 its for testing(use the testing_data matrix)
                                                  // 1 its for adding noise
}
fclose(fin900);

/****************** Initialize Weights ********************/
InitializeWeights();

// ********************** TRAINING ************************** //
int network[units];
int iter = 0;
while (iter<50){
        PositivePhase(network);
        NegativePhase(network);
        iter++;
        tout<<"iter "<<iter<<endl;
        ofstream rout("Recovered Points.txt");
        for (int j=0;j<TEST_PAT;j++)
              ConvergenceFunc(noisy_data[j],j,testing_data[j],rout);
        rout.close();

        /********************** CALCULATE ERROR ********************/
        FILE *err1;
```

```
err1 = fopen("Initial Points.txt","r");
if (err1==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}
FILE *err2;
err2 = fopen("Recovered Points.txt","r");
if (err2==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}
FILE *err3;
err3 = fopen("Initial Points.txt","r");
if (err3==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}
FILE *err4;
err4 = fopen("Recovered Points.txt","r");
if (err4==NULL){
      cout <<"This file doesn't exist"<<"\n";
      exit(-1);
}
FILE *restr;
if (cas>=1 && cas<=6){
      restr = fopen("Initial Occluded Points.txt","r");
      if (restr==NULL){
            cout <<"This file doesn't exist"<<"\n";
            exit(-1);
      }
}
FILE *restr2;
if (cas>=1 && cas<=6){
      restr2 = fopen("Initial Occluded Points.txt","r");
      if (restr2==NULL){
            cout <<"This file doesn't exist"<<"\n";
            exit(-1);
      }
```

```
}
char pattern[15];
int pnum;
int pat;
int count_pat;
int x1,y1,x2,y2;
int x3,y3;
double error,rerror,o_deviation,r_deviation;
double total_error,total_rerror;
double od_mean,rd_mean;
vector<double> r_di,o_di;
int if_occluded;
int count_occ =0;
double o_sum,r_sum;
total_error = 0;
total_rerror = 0;
count_pat = 0;
o_sum = 0.0;
r_sum = 0.0;
o_deviation =0;
r_deviation = 0;

while ( (!feof(err1)) && count_pat<900){

	count_pat++;
	fscanf(err1,"%s",pattern);    // read the word Pattern
	fscanf(err2,"%s",pattern);    // read the word Pattern
	fscanf(err1,"%d",&pnum);      // read the number next to Pattern
	fscanf(err2,"%d",&pnum);      // read the number next to Pattern
	rerror = 0;
	error = 0;
	count_occ =0;
	for(pat=0;pat<68;pat++){
		// read from "Initial Points.txt" x1 and y1
		fscanf(err1,"%d",&x1);
		fscanf(err1,"%d",&y1);
		// read from "Recovered Points.txt" x2 and y2
		fscanf(err2,"%d",&x2);
```

```
            fscanf(err2,"%d",&y2);

            if (cas>=1 && cas<=6){
                  // read from "Initial Occluded Points.txt" x3 and y3
                  fscanf(restr,"%d",&if_occluded);
                  fscanf(restr,"%d",&x3);
                  fscanf(restr,"%d",&y3);
                  if (if_occluded==1){
                        count_occ++;
                        rerror = rerror + pow(double(x2-x3),2) + pow(double(y2-y3),2);
                  }
            }
            error = error + (pow(double(x2-x1),2) + pow(double(y2-y1),2));
      }
      error = (double)sqrt((double)error);
      error = (double)error/(double)68.0;
      total_error = total_error + error; //total overall error

      if (cas>=1 && cas<=6){
            rerror = (double)sqrt((double)rerror);
            rerror = (double)rerror/(double)68.0;
            total_rerror = total_rerror + rerror; //total restricted error
      }
}
fclose(err1);
fclose(err2);
if (cas>=1 && cas<=6)
      fclose(restr);
eout<<"iter "<<iter<<endl;
total_error = (double)total_error/(double)count_pat;
eout<<"Mean Overall Error = "<<total_error<<endl;
count_pat = 0;
count_occ=0;

while ( (!feof(err3)) && count_pat<900 ){
      count_pat++;
      fscanf(err3,"%s",pattern);          // read the word Pattern
      fscanf(err4,"%s",pattern);          // read the word Pattern
```

```
	fscanf(err3,"%d",&pnum);			// read the number next to Pattern
	fscanf(err4,"%d",&pnum);			// read the number next to Pattern
	// calculate standard deviation of overall error
	for(pat=0;pat<68;pat++){
		// read from "Initial Points.txt" x1 and y1
		fscanf(err3,"%d",&x1);
		fscanf(err3,"%d",&y1);
		// read from "Recovered Points.txt" x2 and y2
		fscanf(err4,"%d",&x2);
		fscanf(err4,"%d",&y2);
		o_sum = o_sum + sqrt((double)(pow(double(x2-x1),2) + pow(double(y2-y1),2)));

		if (cas>=1 && cas<=6){
			fscanf(restr2,"%d",&if_occluded);
			fscanf(restr2,"%d",&x3);
			fscanf(restr2,"%d",&y3);
			if (if_occluded==1){
				count_occ++;
				r_sum = r_sum + sqrt(pow(double(x2-x3),2) + pow(double(y2-y3),2));

			}
		}
	}
	o_sum = (double)o_sum/(double)68.0;
	o_di.push_back(o_sum);

	if (cas>=1 && cas<=6){
		r_sum = (double)r_sum/(double)68.0;
		r_di.push_back(r_sum);
	}
	o_sum = 0;
	r_sum = 0;
	count_occ = 0;
}
od_mean = 0.0;
rd_mean = 0.0;
for (int f=0;f<o_di.size();f++){
	od_mean = od_mean + o_di[f];
```

```
                if (cas>=1 && cas<=6)
                        rd_mean = rd_mean + r_di[f];
        }
        od_mean = od_mean / (double)count_pat;
        if (cas>=1 && cas<=6)
                rd_mean = rd_mean / (double)count_pat;

        for (int f=0;f<o_di.size();f++){
                o_deviation = o_deviation + (o_di[f] - od_mean)* (o_di[f] - od_mean);
                if (cas>=1 && cas<=6)
                        r_deviation = r_deviation + (r_di[f] - rd_mean)* (r_di[f] - rd_mean);
        }
        o_deviation = o_deviation/(double)count_pat;
        o_deviation = sqrt(o_deviation);

        if (cas>=1 && cas<=6){
                r_deviation = r_deviation/(double)count_pat;
                r_deviation = sqrt(r_deviation);
        }
        dout<<"iter "<<iter<<endl;
        dout<<"Standard Deviation of Overall Error = "<<o_deviation<<endl;

        if (cas>=1 && cas<=6){
                total_rerror = (double)total_rerror/(double)count_pat;
                eout<<"Mean Restricted Error = "<<total_rerror<<endl;
                dout<<"Standard Deviation of Restricted Error = "<<r_deviation<<endl;
        }
        fclose(err3);
        fclose(err4);
        if (cas>=1 && cas<=6)
                fclose(restr2);
    }
    cout<<endl<<"Exiting..."<<endl;
    return 0;
}
```