ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Πτυχιακή Εργασία

Μηχανική Εκμάθηση σε Ασύρματα Δίκτυα Αισθητήρων

Παπαδόπουλος Μιχάλης

Α.Μ: 2011524

Επιβλέποντας: Κ. Ευστάθιος Χατζηευθυμιάδης (Επίκουρος Καθηγητής)

ΑΘΗΝΑ

ΟΚΤΩΒΡΙΟΣ 2015

ΠΕΡΙΛΗΨΗ

Ο έγκαιρος εντοπισμός πυρκαγιάς σε δασικές περιοχές, μπορεί να προστατέψει ανθρώπινες ζωές και πολύτιμες δασικές εκτάσεις, απαραίτητες για τη διατήρηση της ισορροπίας του περιβάλλοντος και τη διαβίωση των ζώων που κατοικούν σε αυτές. Σε αυτή την εργασία εξετάζεται η υλοποίηση ενός ασύρματου δικτύου αισθητήρων, με το οποίο συλλέγονται μετρήσεις για τη θερμοκρασία και την υγρασία του περιβάλλοντος χώρου προκειμένου στη συνέχεια αυτές να τροφοδοτειθούν σε ένα νευρωνικό δίκτυο με στόχο την εξαγωγή συμπεράσματος για την ύπαρξη ή μη, φωτιάς. Κύριο στόχο της εργασίας αυτής αποτελεί η διερεύνηση του εάν μπορεί να υλοποιηθεί ένα νευρωνικό δίκτυο, υποστηριζόμενο από τη γλώσσα ανάπτυξης (j2me) της πλατφόρμας Sunspot της Oracle, προκειμένου να παρέχεται έγκαιρη και έγκυρη πληροφόρηση σχετικά με την ύπαρξη ή μη φωτιάς σε δασικές εκτάσεις.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Εκμάθηση σε Ασύρματα Δίκτυα Αισθητήρων. Υλοποίηση νευρωνικού δικτύου σε δίκτυο αισθητήρων για ανίχνευση πυρκαγιάς.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Σύντηξη δεδομένων, νευρωνικά δίκτυα, δίκτυα αισθητήρων, moving average

ABSTRACT

Well-timed fire detection in forests can protect human lives and valuable forest areas, necessary for the maintenance of environmental equilibrium and for the living of animals residing there. In the present thesis we examine the implementation of a wireless sensor network which gathers temperature and humidity data from the surrounding environment. The data then are fed to a neural network which is implemented on one of the network's nodes in order to obtain a valid conclusion as to whether there is fire activity or not. The main goal of this thesis is to investigate whether a neural network, supported by Oracle's SUNSpot platform development programming language (j2me), can be implemented in order to provide valid and time effective information as to whethere a forest fire has bursted out or not.

SUBJECT AREA: Machine Learning on Wireless Sensor Networks. Neural network implementation on a wireless sensor network for fire detection.

KEYWORDS: Data fusion, Neural Networks, wireless sensor networks, moving average

ΕΥΧΑΡΙΣΤΙΕΣ

Για την διεκπεραίωση της παρούσας εργασίας θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή της διπλωματικής μου εργασίας κύριο Χατζηευθυμιάδη Ευστάθιο για την καθοδήγηση του, την εμπιστοσύνη του και την ευκαιρία που μου έδωσε να ασχοληθώ με ένα τόσο ενδιαφέρον θέμα και τον κύριο Ρείση Διονύση για την επιμονή του και την διδασκαλία του μέσα από την οποία απέκτησα γνώσεις που με βοήθησαν να ολοκληρώσω την εργασία. Επίσης θα ήθελα να ευχαριστήσω τους κύριους Sean Barbeau (encogME) και Roar Lauritzsen (Real.java) για την τεχνική υποστήριξη τους Τέλος θα ήθελα να ευχαριστήσω θερμά τους: Παπαδοπούλου Θεοδώρα, Παπαδόπουλο Ιωάννη, Καραφυλή Νίκο και Χρυσανθοπούλου Νεφέλη για την υπομονή και την υποστήριξη τους.

Περιεχόμενα.

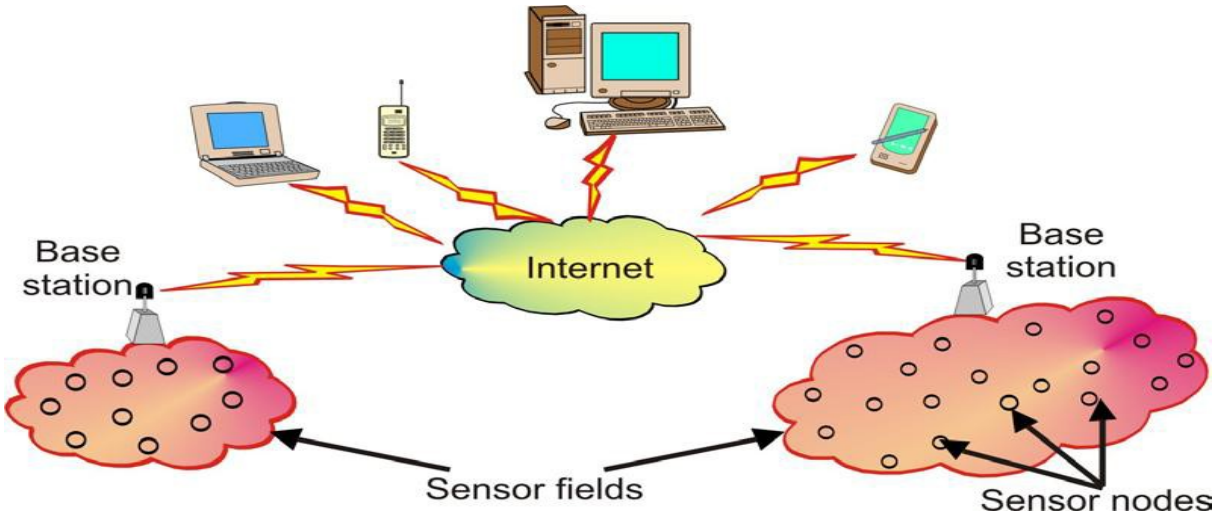
1. Ασύρματα Δίκτυα Αισθητήρων. Μηχανική Εκμάθηση

Ασύρματο δίκτυο αισθητήρων (Wireless Sensor Network: WSN) ορίζεται ένα ασύρματο δίκτυο από αυτόνομες συσκευές κατανεμημένες στον χώρο που χρησιμοποιούν αισθητήρες για να μετράνε φυσικά ή περιβαλλοντικά μεγέθη. Μέχρι στιγμής τα WSN έχουν χρησιμοποιηθεί σε τομείς όπως ο υγείας, διαχείρισης ενέργειας, απομακρυσμένης επίβλεψης. Στον τομέα υγείας τα WSN μπορούν να προσφέρουν περισσότερο διακριτική παρακολούθηση ασθενούς και καλύτερη εφαρμογή θεραπείας. Στον τομέα της διαχείρισης ενέργειας, όπως το δίκτυο διανομής ηλεκτρικής ενέργειας, το δίκτυο δημοτικού φωτισμού και ύδρευσης, τα WSN μπορούν να προσφέρουν μια χαμηλού κόστους μέθοδο συλλογής δεδομένων κατάστασης/υγείας του συστήματος με σκοπό την καλύτερη διαχείριση ενέργειας και πόρων. Οι εφαρμογές απομακρυσμένης επίβλεψης περιλαμβάνουν:

- Παρακολούθηση αέρα, νερού και εδάφους του περιβάλλοντος
- Παρακολούθηση της δομικής ακεραιότητας κτηρίων και γεφυρών
- Παρακολούθηση λειτουργίας βιομηχανικών μηχανημάτων σε γραμμές παραγωγής
- Παρακολούθηση διαδικασιών
- Παρακολούθηση πορείας φυσικών αντικειμένων είτε χρησιμοποιώντας barcode ετικέτες που φέρουν τα φυσικά αντικείμενα είτε με χρήση GPS ή RFID τεχνολογίας, εάν τα φυσικά αντικείμενα χρησιμοποιούν την τεχνολογία αυτή.

1.1 Αρχιτεκτονική δικτύου Αισθητήρων

Ένα ασύρματο δίκτυο αισθητήρων αποτελείται από έναν μεγάλο αριθμό αισθητήρων κατανεμημένων ομοιόμορφα σε περιορισμένη γεωγραφική περιοχή. Οι κόμβοι-αισθητήρες (sensor nodes) διασπείρονται στο πεδίο παρατήρησης (sensor field) με τον τρόπο που φαίνετε στην Εικόνα 1.



Εικόνα 1. Ασυρματο Δικτυο Αισθητηρων

Κάθε ένας από τους διασκορπισμένους κόμβους έχει την ικανότητα να συλλέγει δεδομένα και να τα δρομολογεί στο κέντρο συλλογής, σε έναν κόμβο-πηγή (base station ή sink). Το κέντρο συλλογής είναι δυνατόν να επικοινωνεί με τον κόμβο διαχείρισης της εφαρμογής (task manager node) μέσω του Διαδικτύου ή μέσω δορυφόρου

1.3 Μηχανική Εκμάθηση Σε Ασύρματα Δίκτυα Αισθητήρων.

Συνήθως τα ασύρματα δίκτυα αισθητήρων παρακολουθούν περιβάλλοντα που μεταβάλλονται δυναμικά στην πορεία του χρόνου. Δεδομένου ότι υπάρχει ανάγκη ένα WSN να προσαρμόζεται στις μεταβολές του περιβάλλοντος το οποίο παρακολουθεί, πολλές φορές χρησιμοποιούνται τεχνικές μηχανικής εκμάθησης τόσο στον σχεδιασμό όσο και στην λειτουργία του WSN προκειμένου να αποφευχθεί η ανάγκη για επανασχεδιασμό του συστήματος.

Μηχανική εκμάθηση, σύμφωνα με δυο κλασικούς ορισμούς της, είναι :

1. Η ανάπτυξη μοντέλων υπολογιστών με σκοπό την εκμάθηση διαδικασιών που προσφέρουν λύσεις στο πρόβλημα απόκτησης γνώσης και επαυξάνουν την απόδοση ήδη αναπτυγμένων συστημάτων.
2. Η υιοθέτηση υπολογιστικών μεθόδων με σκοπό την βελτίωση της απόδοσης μηχανών, ανιχνεύοντας συνοχές και patterns σε δεδομένα εκπαίδευσης.

Εφαρμόζοντας τους δυο ορισμούς στα WSN, η μηχανική εκμάθηση ερμηνεύεται ως η εκμετάλλευση δεδομένων προηγούμενου χρόνου, με σκοπό την βελτίωση της απόδοσης των

WSN χωρίς επαναπρογραμματισμό. Συγκεκριμένα η μηχανική εκμάθηση είναι ένα σημαντικό εργαλείο για ένα WSN γιατί μπορεί να προσφέρει λύση στους εξής τομείς

1. Γεωγραφική θέση αισθητήρων. Σε δυναμικά μεταβαλλόμενα περιβάλλοντα μπορεί η γεωγραφική θέση ενός αισθητήρα να αλλάξει και είναι θεμιτό ένα WSN να είναι ικανό να προσαρμοστεί και να λειτουργήσει σε τέτοιες δυναμικές μεταβολές
2. Σε δυναμικά μεταβαλλόμενα περιβάλλοντα όπως ηφαίστεια, οι μεταβολές και οι στάθμες των τιμών των υπό παρακολούθηση μεγεθών, δεν μπορούν να προβλεφθούν με ακρίβεια. Μέσω της μηχανικής εκμάθησης ένα σύστημα μπορεί να είναι ικανό να επαναβαθμονομηθεί εκ νέου βάσει νέων δεδομένων.
3. Οι σχεδιαστές WSN συστημάτων έχουν συχνά στην κατοχή τους μεγάλα σετ δεδομένων προηγούμενου χρόνου, όμως αδυνατούν να διακρίνουν πιθανές συσχετίσεις που υπάρχουν, μεταξύ των δεδομένων. Με κάποια τεχνική μηχανικής εκμάθησης είναι δυνατή η εύρεση των συσχετίσεων μεταξύ των δεδομένων.

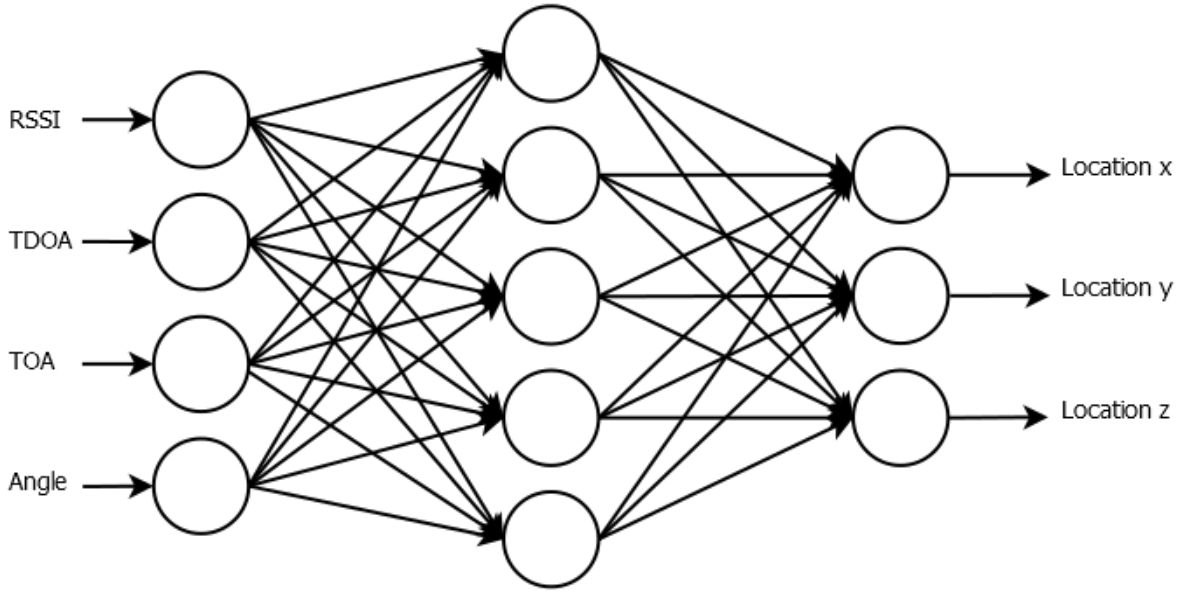
Αν και η μηχανική εκμάθηση σαν εργαλείο μπορεί να προσφέρει λύση σε αρκετά προβλήματα που συχνά προκύπτουν σε WSN, υπάρχουν κάποια μειονεκτήματα και περιορισμοί στην χρήση του:

1. Ως σύστημα περιορισμένων πόρων ένα WSN καταναλώνει σημαντικό ποσοστό αυτών στην πρόβλεψη και εξαγωγή μια ομόφωνης σχέσης ανάμεσα στα δεδομένα. Επομένως η αναλογία του υπολογιστικού κόστους του μοντέλου με την ακρίβεια του μοντέλου είναι κάτι που πρέπει να λαμβάνεται υπ όψιν του σχεδιαστή του συστήματος, δεδομένου ότι μεγαλύτερη ακρίβεια οδηγεί σε μεγαλύτερη ενεργειακή κατανάλωση.
2. Η εκμάθηση μέσω παραδειγμάτων απαιτεί μεγάλα σετ δεδομένων, προκειμένου να επιτευχτεί το επιθυμητό όριο σφάλματος και παράλληλα ο σχεδιαστής έχει περιορισμένο έλεγχο στην διαδικασία της εκμάθησης.

Οι περισσότεροι υπάρχοντες αλγόριθμοι μηχανικής εκμάθησης κατηγοριοποιούνται βάσει του τρόπου με τον οποίο εκπαιδεύονται και ανήκουν σε μια από τις κατηγορίες της Εποπτευόμενης, Μη Εποπτευόμενης και Ενισχυτικής εκμάθησης. Συγκεκριμένα, η εποπτευόμενη εκμάθηση είναι η διαδικασία κατά την οποία ο αλγόριθμος μηχανικής εκμάθησης τροφοδοτείται με δεδομένα εκπαίδευσης τα οποία αποτελούνται από εισόδους και τις αντίστοιχες αναμενόμενες εξόδους. Το μοντέλο που υλοποιείται αναπαριστά την σχέση ανάμεσα στις παρεχόμενες τιμές εισόδου και τις

παρεχόμενες τιμές εξόδου. Αντίθετα στην μη εποπτευόμενη εκμάθηση τα δεδομένα που τροφοδοτούνται στον αλγόριθμο εκμάθησης δεν περιέχουν αναμενόμενη έξοδο και στόχος του αλγορίθμου είναι η κατηγοριοποίηση δειγμάτων των δεδομένων εκπαίδευσης ανιχνεύοντας ομοιότητες μεταξύ των δεδομένων αυτών. Στην τελευταία μεγάλη κατηγορία, την κατηγορία ενισχυτικής εκμάθησης, ο αλγόριθμος εκμάθησης εκπαιδεύεται αλληλεπιδρώντας με το περιβάλλον του (online training).

Στην κατηγορία της Εποπτευόμενης εκμάθησης, ένα σετ δεδομένων με προκαθορισμένες εισόδους και γνωστές εξόδους χρησιμοποιείται για την εκπαίδευση του μοντέλου. Κατόπιν εκπαίδευσης το μοντέλο πλέον αναπαριστά την σχέση μεταξύ εισόδου και εξόδου. Σε αυτή την κατηγορία, τα νευρωνικά δίκτυα αποτελούν έναν αλγόριθμο μηχανικής εκμάθησης που μπορεί εύκολα να δώσει λύσεις σε προβλήματα που συναντώνται συχνά σε WSN. Για παράδειγμα το πρόβλημα εντοπισμού γεωγραφικής θέσης κόμβου μπορεί να λυθεί εύκολα, εάν είναι διαθέσιμες μετρήσεις γωνίας και απόστασης σημάτων που λαμβάνονται σε κόμβους τοποθετημένους σε στρατηγικά σημεία (anchor nodes). Οι μετρήσεις απόστασης μπορεί να περιλαμβάνουν τον δείκτη ισχύος σήματος (received signal strength indicator - RSSI), την ώρα άφιξης του σήματος στον anchor node (time of arrival - TOA) και την χρονική διαφορά άφιξης του σήματος (time difference of arrival - TDOA) όπως φαίνετε στην Εικόνα 2.



Εικόνα 2. Νευρωνικό δίκτυο που λύνει το πρόβλημα εντοπισμού γεωγραφικής θέσης κόμβου.

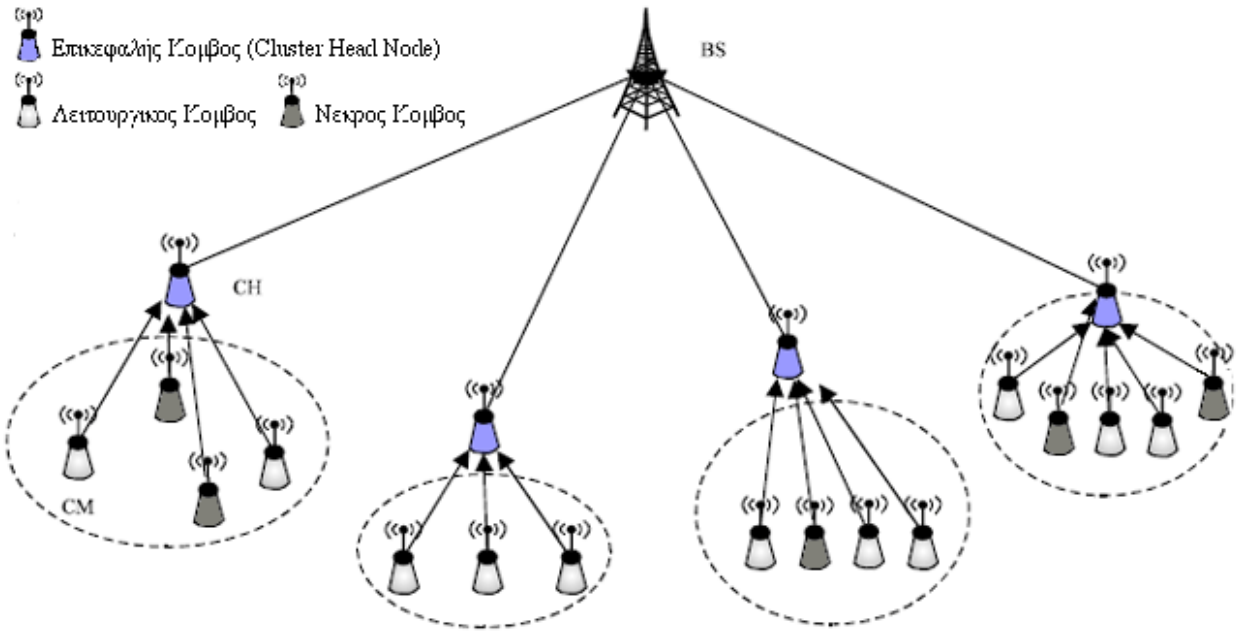
Μετά την εποπτευόμενη εκπαίδευση το νευρωνικό δίκτυο παράγει ένα διάνυσμα που περιέχει μια εκτίμηση των συντεταγμένων του κόμβου, στον τρισδιάστατο χώρο.

1.4. Λειτουργικές Προκλήσεις

Κατά τον σχεδιασμό ενός WSN, παράγοντες όπως η περιορισμένη μνήμη και το περιορισμένο ενεργειακό απόθεμα των κόμβων που λαμβάνουν μετρήσεις, τοπογραφικές αλλαγές και μια πιθανή αποτυχία ζεύξης επικοινωνίας, είναι σημαντικό να λαμβάνονται υπ όψιν προκειμένου να σχεδιαστεί ένα εύρωστο λειτουργικό σύστημα. Αν και υπάρχει πληθώρα αλγορίθμων μηχανικής εκμάθησης που μπορεί να λύσει τα περισσότερα από αυτά τα προβλήματα, στην παρούσα εργασία θα ασχοληθούμε με τα προβλήματα ενός WSN που απαντώνται με την χρήση νευρωνικών δικτύων.

1.4.1. Ομαδοποίηση και Συσσωμάτωση Δεδομένων (Clustering and Data Aggregation)

Σε μεγάλης κλίμακας WSN, περιορισμένου ενεργειακού αποθέματος, είναι ασύμφορο όλα τα δεδομένα να μεταδίδονται συνεχώς προς τον κεντρικό κόμβο. Μια αποτελεσματική λύση είναι η ομαδοποίηση των κόμβων σε μικρότερα σύνολα (Clusters). Κάθε cluster έχει έναν επικεφαλή κόμβο (cluster head ή aggregator). Ο επικεφαλής κόμβος συσσωρεύει τα δεδομένα από όλους τους κόμβους που ανήκουν στο cluster του και τα προωθεί στο κεντρικό κόμβο. Συνήθως αυτό έχει ως αποτέλεσμα την εξοικονόμηση ενέργειας.



Εικόνα 3. Ομαδοποιημένη αρχιτεκτονική με ενεργούς και νεκρούς κόμβους.

Στην Εικόνα 3 αναπαριστάτε η συσσώρευση δεδομένων, από κόμβους αισθητήρων σε Aggregator κόμβους και από εκεί σε κεντρικούς Sink κόμβους. Σε ένα τέτοιο σενάριο μπορεί να υπάρξουν κόμβοι αισθητήρων που αποστέλλουν λανθασμένες μετρήσεις. Οι μετρήσεις από αυτούς τους κόμβους μπορούν να επηρεάσουν αρνητικά την ολική ακρίβεια του συστήματος επομένως πρέπει να απομακρυνθούν από το δίκτυο. Τεχνικές μηχανικής εκμάθησης όπως το νευρωνικό δίκτυο μπορούν να χρησιμοποιηθούν για να λύσουν θέματα που προκύπτουν σε ένα τέτοιο σενάριο ως εξής:

- Χρήση νευρωνικού δικτύου για τοπική (στον επικεφαλή του cluster) συμπίεση δεδομένων, αφαιρώντας από το σύνολο των μετρήσεων ανομοιότητες ανάμεσα στις μετρήσεις κόμβων αισθητήρων.
- Χρήση νευρωνικού δικτύου για την επιλογή κατάλληλου Επικεφαλή Cluster, όπου η σωστή επιλογή κατάλληλου Επικεφαλή μπορεί να αποφέρει εξοικονόμηση ενέργειας και να επιμηκύνει τον χρόνο ζωής του δικτύου.

1.4.2. Ανίχνευση Συμβάντων

Μια βασική απαίτηση από ένα WSN είτε είναι μεγάλης είτε μικρής κλίμακας είναι η ανίχνευση γεγονότων. Συνεπώς ένα WSN για να είναι ικανό να παρέχει τέτοια υπηρεσία πρέπει να έχει αξιόπιστο προγραμματισμό γεγονότων (event scheduler) και ανίχνευση γεγονότων με την ελάχιστη δυνατή ανθρώπινη (προγραμματιστική) παρέμβαση. Η παρακολούθηση μεγεθών στα WSN μπορεί να είναι είτε καθοδηγούμενη από κάποιο γεγονός (event driven), είτε συνεχής, είτε καθοδηγούμενη από επερώτηση χρήστη. Οι απλούστερες μέθοδοι ανίχνευσης γεγονότος βασίζονται στην ρύθμιση κάποιας αυστηρά ορισμένης τιμής κατωφλίου, με την παραγωγή συναγερμού κάθε φορά που το υπό παρακολούθηση μέγεθος ξεπερνά την τιμή αυτή. Με την χρήση νευρωνικού δικτύου μπορεί να υλοποιηθεί σύστημα το οποίο εκτός από την τιμή κατωφλίου, χρησιμοποιεί πληροφορία που προκύπτει από την επεξεργασία των μετρήσεων εισόδου. Για παράδειγμα, νευρωνικά δίκτυα χρησιμοποιήθηκαν ενεργά σε συστήματα ανίχνευσης πυρκαγιάς επιτυγχάνοντας καλύτερα αποτελέσματα από συστήματα που βασίζονται σε δορυφόρους, ενώ παράλληλα κοστίζουν πολύ λιγότερο. Οι Yu et al [1], έχουν παρουσιάσει σύστημα ανίχνευσης πυρκαγιάς βασισμένο σε νευρωνικό δίκτυο στο οποίο η επεξεργασία δεδομένων κατανείμετε στους Επικεφαλείς των Clusters και μόνο η πληροφορία που κρίνεται ως χρήσιμη προωθείται από τους Επικεφαλείς σε κάποιο κεντρικό σταθμό (Basestation).

1.4.3. Γεωγραφικός Εντοπισμός (Localization) και Στόχευση Αντικειμένων

Ο Γεωγραφικός Εντοπισμός είναι η διαδικασία του προσδιορισμού της γεωγραφικής θέσης ενός κόμβου ή αντικειμένου του WSN. Η επίγνωση της θέσης των κόμβων αισθητήρων του δικτύου είναι σημαντική παράμετρος, δεδομένου ότι τα περισσότερα WSN χρησιμοποιούν την πληροφορία αυτή, κατά την λειτουργία τους. Στα περισσότερα σύστημα, η χρήση GPS τεχνολογίας είναι οικονομικά ασύμφορη και σε πολλές περιπτώσεις ακόμα και αδύνατη - για παράδειγμα σε σύστημα εσωτερικού χώρου. Παράλληλα η σχετική θέση ενός κόμβου είναι χρήσιμη σε ένα περιορισμένο φάσμα εφαρμογών. Αντίθετα η χρήση της απόλυτης θέσης ενός μικρού συνόλου κόμβων (Anchor Nodes ή Beacon Node: Κόμβος του οποίου η απόλυτη θέση είναι γνωστή καθώς ο κόμβος είναι εξοπλισμένος με τεχνολογία GPS) μπορεί να χρησιμοποιηθεί ώστε να μετατραπούν οι σχετικές θέσεις των υπολοίπων κόμβων του δικτύου που δεν αποτελούν Anchor Nodes, σε απόλυτες θέσεις. Προκειμένου να ενισχυθεί η απόδοση του συστήματος εντοπισμού θέσης μέσω κόμβων Anchor, χρησιμοποιούνται μετρήσεις όπως οι RSSI, TOA και TDOA. Επιπλέον η γωνία του σήματος μπορεί να μετρηθεί με την χρήση πυξίδας η έξυπνης κεραίας. Οι Shareef et al [1], συνέκριναν τρεις διαφορετικές υλοποιήσεις γεωγραφικού εντοπισμού βασισμένες σε νευρωνικά δίκτυα. Συγκεκριμένα η μελέτη, εστιάζει σε εντοπισμό θέσης κόμβου σε WSN με χρήση Multi-Layer Perceptron (MLP), Radial Basis Function (RBF) και recurrent Neural Networks (RNN). Από τις τρείς υλοποιήσεις, η RBF παρουσίασε το μικρότερο ποσοστό σφάλματος έχοντας το μεγαλύτερο ενεργειακό και επεξεργαστικό κόστος ενώ η MLP υλοποίηση έχει την μικρότερη ενεργειακή και επεξεργαστική κατανάλωση αλλά μεγαλύτερο ποσοστό σφάλματος. Το κύριο πλεονέκτημα των βασισμένων σε νευρωνικό δίκτυο, αλγορίθμων γεωγραφικού εντοπισμού είναι η ικανότητα τους να παρέχουν ως έξοδο, διανύσματα συντεταγμένων στον τρισδιάστατο χώρο.

1.5. Μη Λειτουργικές Προκλήσεις

Οι μη λειτουργικές προκλήσεις περιλαμβάνουν λειτουργίες που δεν σχετίζονται με την βασική λειτουργική συμπεριφορά ενός WSN. Μια τέτοια λειτουργία η οποία μπορεί να βρει λύση με την χρήση νευρωνικού δικτύου, είναι η εξασφάλιση Ποιότητας Υπηρεσίας (Quality of Service - QoS). Σε ένα WSN υπάρχει πιθανότητα να χρειάζονται multi hop μεταδόσεις προκειμένου να φτάσει ένα μήνυμα από κάποιο basestation σε κάποιο κόμβο του δικτύου όπως

επίσης υπάρχει η ανάγκη μετάδοσης επερωτήσεων από κάποιο ελεγκτή του συστήματος στους κόμβους του δικτύου. Το γεγονός ότι τα WSNs έχουν περιορισμένο ενεργειακό απόθεμα και εύρος ζώνης οριοθετεί την ποσότητα πληροφορίας που μπορεί να μεταδοθεί από κάποια πηγή του δικτύου προς τους υπόλοιπους κόμβους του δικτύου. Παράλληλα η συσσώρευση και η διάδοση πληροφορίας μπορεί να είναι αναξιόπιστη και το γεγονός αυτό σε συνδυασμό με το ότι η τοπολογία ενός WSN μπορεί να πάρει οποιαδήποτε μορφή αντί μιας καθορισμένης, καθιστούν τον σχεδιασμό αλγορίθμου διαχείρισης ενός WSN αρκετά απαιτητική διαδικασία. Οι Snow et al [1] εισήγαγαν μια μέθοδο για εκτίμηση της αξιοπιστίας του δικτύου, η οποία υλοποιείται με χρήση νευρωνικού δικτύου. Η αξιοπιστία δικτύου είναι μια μονάδα μέτρησης που εκπροσωπεί την διαθεσιμότητα, διατηρησιμότητα και την επιβιωσημότητα του δικτύου και για την εύρεση της χρησιμοποιούνται παράμετροι όπως ο μέσος χρόνος μεταξύ ύπαρξης σφαλμάτων (mean time between failure - MTBF) και ο μέσος χρόνος επιδιόρθωσης (mean time to repair MTTR). Παράλληλα οι Moustapha και Selmic [1] παρουσίασαν ένα δυναμικό μοντέλο ανίχνευσης σφαλμάτων το οποίο υπολογίζει την δυναμική συμπεριφορά του κάθε κόμβου του δικτύου και την επίδραση που έχει αυτός σε άλλους κόμβους. Για την αναγνώριση (από το σύνολο των κόμβων) κόμβου και την ανίχνευση σφαλμάτων χρησιμοποιήθηκε νευρωνικό δίκτυο του οποίο εκπαιδεύτηκε με τον αλγόριθμο εκπαίδευσης ανάστροφης διάδοσης (Backpropagation).

1.6. Συμπέρασμα

Τα ασύρματα δίκτυα αισθητήρων διαφέρουν από τα παραδοσιακά δίκτυα, για παράδειγμα χαρακτηρίζονται από περιορισμένα ενεργειακά αποθέματα και επεξεργαστική ισχύ, συνεπώς χρειάζονται εργαλεία για την επίλυση προβλημάτων που πηγάζουν από τους περιορισμούς αυτούς. Η μηχανική εκμάθηση προσφέρει πληθώρα εργαλείων που λύνουν τα δομικά προβλήματα των WSN και τα καθιστούν ικανά να προσαρμοστούν στις μεταβολές των δυναμικών περιβαλλόντων που παρακολουθούν.

2. Τεχνητά Νευρωνικά Δίκτυα

2.1 Τι είναι ένα Τεχνητό Νευρωνικό Δίκτυο.

Στη γενική μορφή του, ένα Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ), είναι μία μηχανή σχεδιασμένη να μοντελοποιεί τον τρόπο με τον οποίο ο εγκέφαλος εκτελεί μία συγκεκριμένη λειτουργία. Ο απλούστερος ορισμός για το ΤΝΔ παρέχεται από τον εφευρέτη ενός από τους πρώτους νεύρο-υπολογιστές Δρ. Robert Hecht-Nielsen κατά τον οποίο ένα ΤΝΔ είναι:

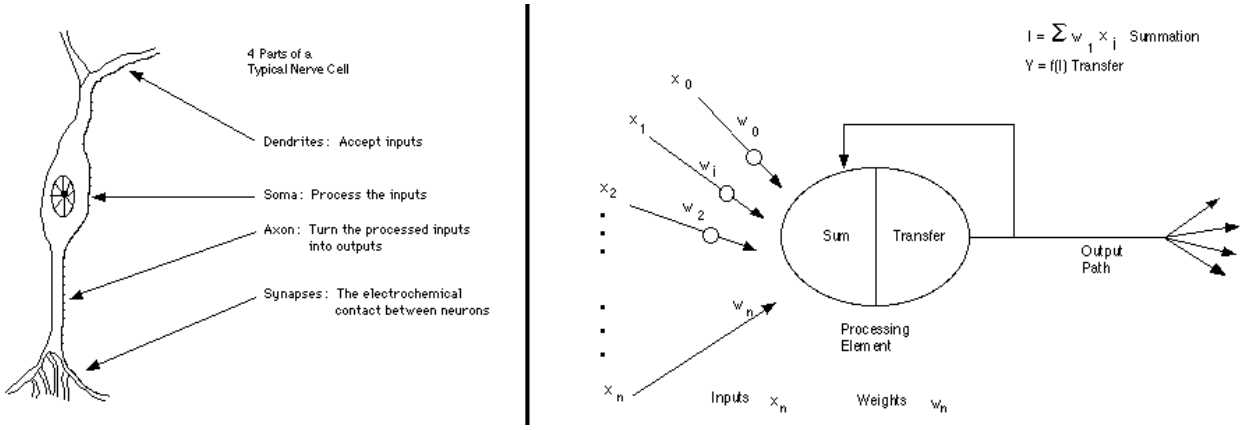
"Ένα υπολογιστικό σύστημα αποτελούμενο από απλά, ιδιαίτερα διασυνδεδεμένα επεξεργαστικά στοιχεία, τα οποία επεξεργάζονται πληροφορία μέσω της δυναμικής απόκρισης κατάστασης σε εξωτερικά δεδομένα εισόδου."[2]

Ένα ΤΝΔ αποτελεί μία μαζικά παραλληλοποιημένη επεξεργαστική μονάδα, συντιθέμενη από απλούς υπολογιστικούς κόμβους, η οποία έχει την φυσική ροπή να αποθηκεύει **πειραματική** γνώση η οποία είναι διαθέσιμη για χρήση. Η λειτουργία των ΤΝΔ προσομοιάζει με αυτή του ανθρώπινου εγκεφάλου στα ακόλουθα δυο σημεία:

1. Το δίκτυο αποκτάει τη γνώση από το περιβάλλον του, μέσω μιας διαδικασίας εκμάθησης.
2. Η αποκτηθείσα γνώση αποθηκεύεται στο δίκτυο με χρήση «Βαρών» στις συνδέσεις μεταξύ των υπολογιστικών κόμβων. Τα βάρη αυτά είναι γνωστά ως συναπτικά βάρη. [3]

Ένα ΤΝΔ μεγάλης κλίμακας μπορεί να αποτελείται από εκατοντάδες ή χιλιάδες υπολογιστικούς κόμβους ενώ ο εγκέφαλος ενός θηλαστικού έχει δισεκατομμύρια υπολογιστικούς κόμβους με μία αντίστοιχη αύξηση σε μέγεθος της συνολικής αλληλεπίδρασης μεταξύ τους. Έτσι, το ΤΝΔ, παρόλο που είναι μοντελοποιημένο βάση της δομής των νευρώνων του εγκεφαλικού φλοιού των θηλαστικών, είναι σημαντικά μικρότερης κλίμακας. Οι υπολογιστικοί κόμβοι που αποτελούν ένα νευρωνικό δίκτυο, ΤΝΔ είτε βιολογικό, ονομάζονται νευρώνες και σε αυτούς εκτελείται η επεξεργασία της πληροφορίας με λήψη και αποστολή σημάτων από και προς άλλους νευρώνες. Η διαδικασία εκμάθησης αποτελείται από παρουσίαση στο δίκτυο εισόδων και των αντίστοιχων γνωστών εξόδων που αναμένονται για τις δεδομένες εισόδους. Όπως και στα βιολογικά νευρωνικά δίκτυα, ένα ΤΝΔ, κατά την εκμάθηση αναπροσαρμόζει τα συναπτικά βάρη που υπάρχουν ανάμεσα στους νευρώνες του, προκειμένου να επιτύχει τη σύγκλιση της ιδεατής

εξόδου (η ήδη γνωστή και αναμενόμενη έξοδος) με την πραγματική (την έξοδο που τελικά διαθέτει το νευρωνικό δίκτυο), κατά το ποσοστό που έχει τεθεί ως στόχος. [4]

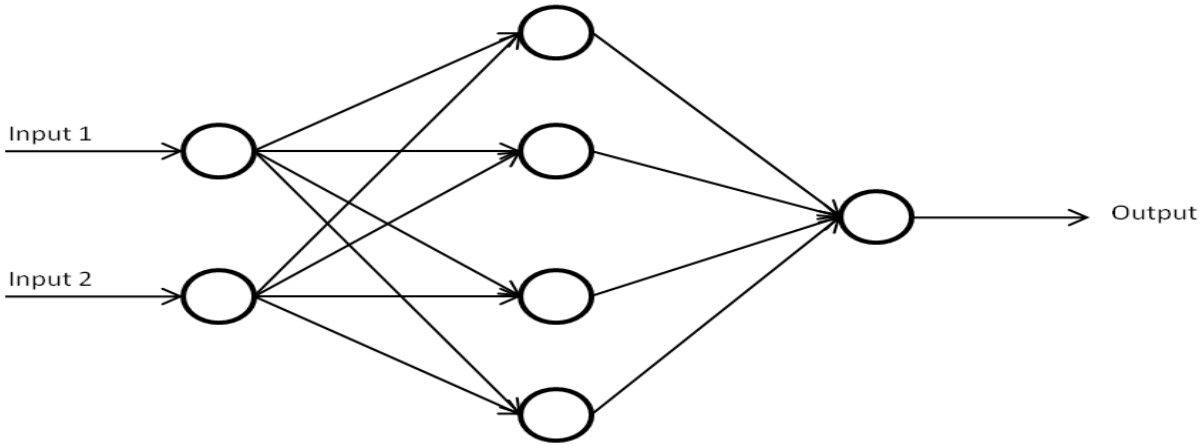


Εικόνα 4: Σύγκριση δομών βιολογικού / τεχνητού νευρώνα. [5]

Στην Εικόνα 4, εμφανίζεται η δομή ενός βιολογικού νευρώνα (αριστερά) σε σύγκριση με ένα νευρώνα ΤΝΔ. Ο βιολογικός νευρώνας δέχεται εισόδους από τους δενδρίτες (Dendrites) του, επεξεργάζεται την πληροφορία στον πυρήνα (Soma) του και παράγει την έξοδο του στο νευρωάξονα (Axon) του, ο οποίος στη συνέχεια διακλαδίζεται σε συνάψεις (Synapses) οι οποίες στην πορεία συνδέονται με δενδρίτες άλλων νευρώνων. Αντίστοιχα ένας τεχνητός νευρώνας δέχεται εισόδους από τους δενδρίτες του $[x_1, x_2, ..., x_n]$, καθένας από τους οποίους είναι έξοδος άλλου νευρώνα ή είσοδος από το περιβάλλον. Στη συνέχεια και αφού γίνει επεξεργασία της πληροφορίας στο επεξεργαστικό κομμάτι του νευρώνα, η έξοδος διοχετεύεται στο μονοπάτι εξόδου (output path – το αντίστοιχο του νευρωάξονα) του νευρώνα, το οποίο είτε αποτελεί την έξοδο του νευρωνικού δικτύου είτε διακλαδίζεται σε συνάψεις οι οποίες συνδέονται σε δενδρίτες νευρώνων επόμενων επιπέδων του νευρωνικού δικτύου και παρέχουν την έξοδο του νευρώνα ως είσοδο στους νευρώνες με τους οποίους συνδέονται. Σημαντικό ρόλο στη λειτουργία του νευρώνα και κατ' επέκταση του νευρωνικού δικτύου παίζουν τα συναπτικά βάρη που υπάρχουν στις συνδέσεις ανάμεσα στους νευρώνες τα οποία, όπως ήδη αναφέρθηκε, αποτελούν το μέσο, μέσω του οποίου το νευρωνικό δίκτυο αποθηκεύει γνώση. Ουσιαστικά το συναπτικό βάρος αποτελεί μέτρο του κατά πόσο μεγάλη επιρροή έχει στην έξοδο του νευρώνα η συγκεκριμένη είσοδος.

Από την άλλη, μία από τις βασικές διαφορές ενός ΤΝΔ με ένα βιολογικό ΝΔ, είναι ότι το ΤΝΔ δεν είναι μία υπολογιστική μονάδα γενικού σκοπού, αλλά περισσότερο ένα εργαλείο επίλυσης ενός πολύ συγκεκριμένου υποπροβλήματος. Έτσι μία ολοκληρωμένη εφαρμογή επίλυσης προβλημάτων συγκεκριμένης φύσεως μπορεί να χρησιμοποίει ένα ή περισσότερα νευρωνικά δίκτυα, το καθένα σχεδιασμένο για την εκτέλεση μίας συγκεκριμένης εργασίας, αλλά δεν μπορεί να αποτελείται εξολοκλήρου από ένα ΤΝΔ.

Ο νευρώνας, λοιπόν, με βάση τα όσα αναφέρονται ανωτέρω, αποτελεί τη δομική μονάδα ενός ΤΝΔ. Στην Εικόνα 5 φαίνεται ο τρόπος με τον οποίο ένα σύνολο νευρώνων διασυνδέονται μεταξύ τους και σχηματίζουν ένα απλό ολοκληρωμένο feed forward νευρωνικό δίκτυο.



Εικόνα 5: Ένα απλό ολοκληρωμένο FeedForward νευρωνικό δίκτυο

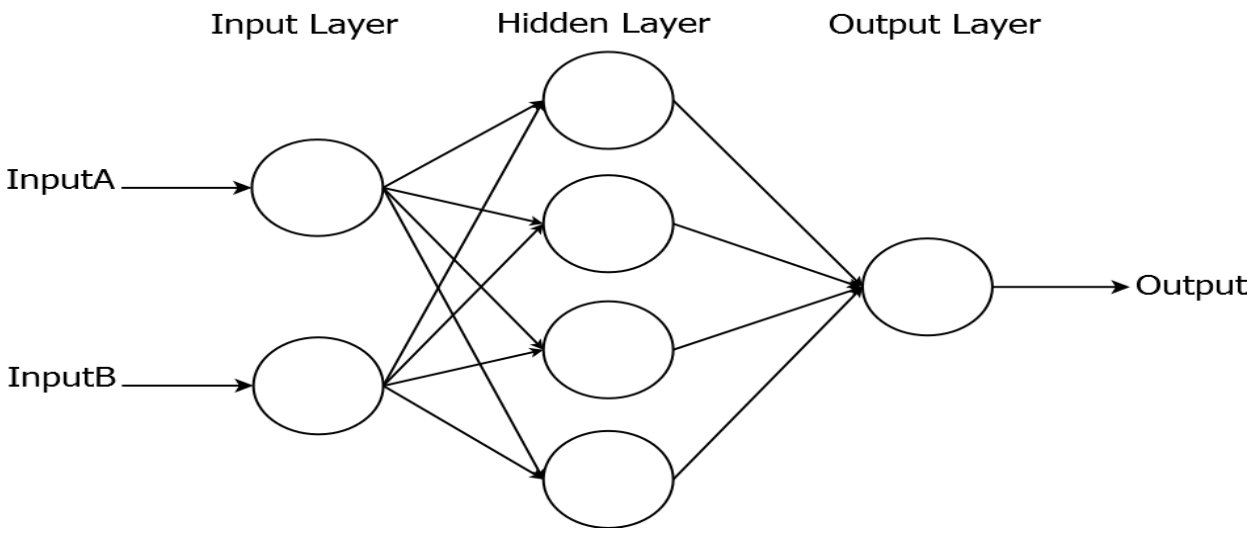
Στην εν λόγω Εικόνα βλέπουμε πως το δίκτυο αποτελείται από ένα σύνολο νευρώνων. Κάθε νευρώνας έχει σαν είσοδο είτε την έξοδο κάποιου νευρώνα από προηγούμενο επίπεδο είτε είσοδο από το περιβάλλον. Οι δυο πρώτοι νευρώνες έχουν εισόδους Input1 και Input2 και η έξοδός τους στη συνέχεια οδηγείται σαν είσοδος στο επόμενο επίπεδο το οποίο αποτελείται από τέσσερις νευρώνες. Τέλος οι έξοδοι των 4 νευρώνων οδηγούνται στην είσοδο του τελευταίου νευρώνα, ο οποίος είναι και αυτός ο οποίος δίνει την τελική έξοδο του δικτύου. Σημειώνεται εδώ ότι στην Εικόνα 5 αποτυπώνεται η αρχιτεκτονική ενός απλού νευρωνικού δικτύου και μόνο. Ο τρόπος λειτουργίας ενός τέτοιου δικτύου και η διαδικασία με την οποία επιλέγεται η αρχιτεκτονική του θα παρουσιαστούν σε επόμενα κεφάλαια.

2.2. Αρχιτεκτονική Νευρωνικού Δικτύου

Αν και η φιλοσοφία πίσω από ένα ΤΝΔ μοιάζει απλή, οι μηχανισμοί που χρησιμοποιούνται ώστε να μπορέσει να λειτουργήσει τελικά ένα νευρωνικό δίκτυο, να αποκτήσει γνώση και αυτή να εφαρμοστεί σε καινούργια απαρουσίαστα δεδομένα, αυξάνουν την πολυπλοκότητα του και αξίζει να μελετηθούν. Διακρίνουμε τρία βασικά επίπεδα ομοίων νευρώνων (κόμβων), τα οποία μπορεί να αποτελούν ένα νευρωνικό δίκτυο (ΤΝΔ):

1. Επίπεδο Εισόδου
2. Κρυφό Επίπεδο
3. Επίπεδο Εξόδου

Στην Εικόνα 6, φαίνεται πως γίνεται ο διαχωρισμός ανάμεσα στα 3 επίπεδα ενός ΤΝΔ.

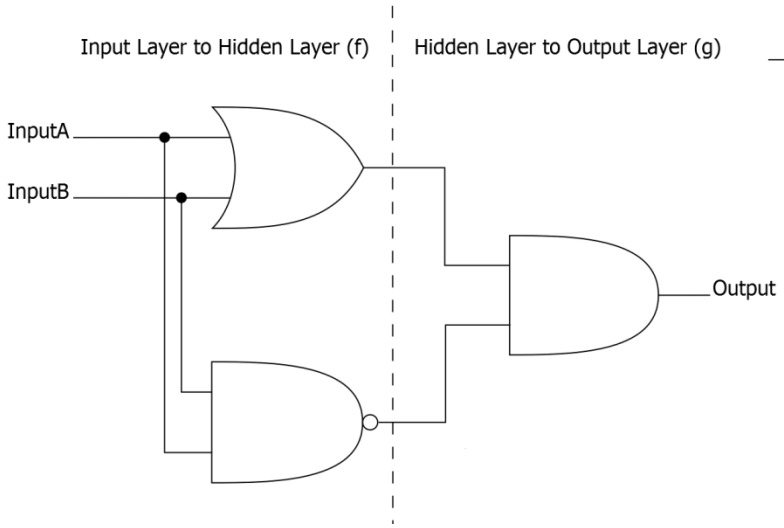


Εικόνα 6. Ο διαχωρισμός των επιπέδων ενός Τεχνητού Νευρωνικού Επιπέδου

Επίπεδο εισόδου. Σε αυτό το επίπεδο οι νευρώνες τροφοδοτούνται στην είσοδο τους από το περιβάλλον και στη συνέχεια απλά μεταβιβάζουν την πληροφορία στο επόμενο, κρυφό, επίπεδο. Για να φτάσουν τα δεδομένα στους κόμβους του κρυφού επιπέδου καμιά επεξεργασία δε λαμβάνει χώρα στο ΤΝΔ. Το πλήθος των κόμβων που αποτελούν το επίπεδο εισόδου συναρτάται με τη φύση των δεδομένων και δεν αποτελεί ελεύθερη επιλογή του σχεδιαστή του δικτύου. . Για παράδειγμα, στο νευρωνικό δίκτυο που αναπτύχθηκε γι' αυτήν την εργασία, τα δεδομένα αποτελούνται από δύο μετρήσεις διαφορετικής φύσης. Μία μέτρηση υγρασίας και μία

θερμοκρασίας. Επομένως, το επίπεδο εισόδου για το συγκεκριμένο ΤΝΔ αποτελείται από δύο κόμβους. Ο πρώτος δέχεται στην είσοδό του τη μέτρηση της θερμοκρασίας και ο δεύτερος αυτή της υγρασίας. Εάν επιθυμούσαμε να κατασκευάσουμε ένα νευρωνικό δίκτυο που μοντελοποιεί τη λογική πράξη XOR τεσσάρων εισόδων, τότε το επίπεδο εισόδου θα αποτελούταν από τέσσερις κόμβους.

Κρυφό Επίπεδο. Η χρήση του κρυφού επίπεδου, αναδεικνύει χαρακτηριστικά και εξαρτήσεις των παραμέτρων της εισόδου. Με αυτόν τον τρόπο επιτρέπεται στο δίκτυο να αναπαραστήσει πολυπλοκότερα μοντέλα. Προκειμένου να γίνει κατανοητό πώς η χρήση ενός κρυφού επιπέδου μπορεί να αυξήσει την ικανότητα του δικτύου να μοντελοποιεί πολυπλοκότερα προβλήματα σκεφτείτε το εξής: Ένα νευρωνικό δίκτυο στην ουσία εφαρμόζει μία σειρά από καθορισμένες συναρτήσεις στα δεδομένα, τα οποία του παρέχονται ως είσοδος. Συνήθως αυτές οι συναρτήσεις είναι περιοριστικές (squashing) συναρτήσεις που εφαρμόζονται στα γραμμικά μετασχηματισμένα δεδομένα, του επιπέδου εισόδου. Ας υποθέσουμε ότι η συνάρτηση από το επίπεδο εισόδου προς το κρυφό επίπεδο είναι η f και η συνάρτηση από το κρυφό επίπεδο προς το επίπεδο εξόδου είναι η g. Ας δεχτούμε, παραδείγματος χάριν, ότι οι συναρτήσεις f και g είναι απλοί λογικοί τελεστές όπως οι AND, OR και NAND. Είναι προφανές ότι χρησιμοποιώντας μόνο μία από τις συναρτήσεις δεν είναι δυνατό να μοντελοποιήσουμε κάποιον άλλο λογικό τελεστή όπως π.χ. τον XOR. Αν όμως εισάγουμε την αρχιτεκτονική επιπέδων και «υλοποιήσουμε» στο πρώτο επίπεδο τους τελεστές NAND και OR και στο δεύτερο επίπεδο τον τελεστή AND τότε, η υλοποίηση ενός σύνθετου τελεστή όπως ο XOR καθίσταται δυνατή.



InpA	InpB	OR	NAND	OR AND NAND	XOR
0	0	0	1	0	0
0	1	1	1	1	1
1	0	1	1	1	1
1	1	1	0	0	0

Εικόνα 7. Μοντελοποίηση τελεστή XOR με χρήση "κρυφού" επιπέδου

Είναι φανερό, λοιπόν, ότι ενώ η κάθε συνάρτηση μόνη της, δεν είναι ικανή να δώσει το επιθυμητό αποτέλεσμα (XOR), ο συνδυασμός τους σε επίπεδα επιτρέπει τη μοντελοποίηση πολυπλοκότερων συναρτήσεων. Με την ίδια φιλοσοφία, ένα νευρωνικό δίκτυο με την προσθήκη κρυφού επιπέδου αποκτάει μεγαλύτερη λειτουργικότητα και μπορεί να αναπαραστήσει πολυπλοκότερα μοντέλα [6].

Το επόμενο και τελευταίο επίπεδο, σε ένα νευρωνικό δίκτυο είναι το επίπεδο εξόδου.

Επίπεδο εξόδου. Το επίπεδο εξόδου, μετά από τη σχετική επεξεργασία των συναρτήσεων που του αντιστοιχούν, είναι αυτό που τελικά μεταδίδει τη ζητούμενη από το δίκτυο πληροφορία στον έξω κόσμο.

Παρόλο που τα επίπεδα - που μπορεί να αποτελούν ένα νευρωνικό δίκτυο - είναι διακριτά και καλά ορισμένα, δεν υπάρχουν ρητοί κανόνες για τη δομή ενός νευρωνικού δικτύου. Έτσι ενώ το πλήθος των επιπέδων που θα αποτελούν το δίκτυο, όπως επίσης και το πλήθος των κόμβων που θα αποτελούν το κάθε κρυφό επίπεδο (εάν αυτό υπάρχει), καθορίζεται από εμπειρικούς κανόνες, ο καθορισμός του πλήθους των κόμβων των επιπέδων εισόδου και εξόδου, υπακούει σε κανόνες που υποδεικνύονται τόσο από τη φύση της εισόδου όσο και από αυτή της επιθυμητής εξόδου. Το απλούστερο δίκτυο αποτελείται από δυο επίπεδα. Το επίπεδο εισόδου και το επίπεδο εξόδου. Τα κρυφά επίπεδα δεν αποτελούν απαραίτητο στοιχείο των νευρωνικών δικτύων, αντιθέτως η χρήση τους ή μη καθορίζεται από τη φύση του εκάστοτε προβλήματος.

Ακολουθούν κάποιοι κανόνες, εμπειρικοί ή και μη εμπειρικοί, για το σχεδιασμό ενός ΤΝΔ:

- Το επίπεδο εισόδου, αποτελείται από τόσους κόμβους όσους επιβάλλει η φύση της εισόδου.
- Η ύπαρξη, η μη, κρυφού επιπέδου εξαρτάται από τη φύση του προβλήματος και των δεδομένων. Ως επί των πλείστον η χρήση κρυφού επιπέδου συνιστάται εάν τα δεδομένα δεν είναι γραμμικά διαχωρίσιμα.
- Το πλήθος τον κόμβων που αποτελούν το κρυφό επίπεδο, συνιστάται να είναι είτε ίσο η κοντά στο μέσο όρο των κόμβων των υπολοίπων επιπέδων, είτε να είναι ίσο με το διπλάσιο των κόμβων του επιπέδου εισόδου. Συνήθως το πλήθος των κόμβων του κρυφού επιπέδου για το κάθε ΤΝΔ, καθορίζεται πειραματικά με δοκιμές και συγκρίσεις αποτελεσμάτων.
- Το πλήθος των κόμβων του επιπέδου εξόδου καθορίζεται από τη μορφή του δικτύου. Εάν το δίκτυο είναι διαμορφωμένο να εκτελεί Regression, τότε αποτελείται από έναν κόμβο, ενώ εάν εκτελεί Classification τότε το επίπεδο εξόδου έχει τόσους κόμβους όσα και τα classes στα οποία καλείται το δίκτυο να κατανείμει τα δεδομένα.
- Οι συναρτήσεις που εφαρμόζονται στα δεδομένα δεν είναι προκαθορισμένες. Υπάρχει ένα πλήθος διαθέσιμων συναρτήσεων – για παράδειγμα η σιγμοειδής συνάρτηση ή η υπερβολική εφαπτομένη - μέσα από το οποίο ο σχεδιαστής καλείται να επιλέξει αυτή που ταιριάζει καλύτερα στα δεδομένα του, δοκιμάζοντας κάθε μία από αυτές και συγκρίνοντας τα αποτελέσματα που λαμβάνει εφαρμόζοντάς τες στο δίκτυο.

Βλέπουμε λοιπόν πως παρόλο που η γενική μορφή ενός ΤΝΔ μπορεί να καθοριστεί εάν είναι γνωστά τα δεδομένα εισόδου - εξόδου, τα τελικά χαρακτηριστικά του, προκύπτουν μόνο μέσα από διαδικασίες δοκιμών με διάφορα πλήθη επιπέδων, διαφορετικές συναρτήσεις ανάμεσα στα επίπεδα και διαφορετικούς τρόπους εκπαίδευσης.

ΣΗΜΕΙΩΣΗ: Regression και Classification είναι δυο από τις λειτουργίες που μπορεί να εκτελέσει ένα ΤΝΔ και θα μελετηθούν με περισσότερη λεπτομέρεια σε επόμενο κεφάλαιο.

2.3. Λειτουργίες Τεχνητού Νευρωνικού Δικτύου

Δυο από τις βασικές λειτουργίες που μπορεί να εκτελέσει ένα ΤΝΔ είναι το Regression και το Classification.

Regression: Σε αυτόν τον τρόπο λειτουργίας, ένα νευρωνικό δίκτυο καλείται να "απαντήσει" στον έξω κόσμο με έναν αριθμό. Σκεφτείτε για παράδειγμα ότι θέλουμε να υπολογίσουμε τα έξοδα αποστολής για μεταφορά πακέτων από ένα σημείο στη χώρα σε κάποιο άλλο. Με την παροχή δεδομένων όπως ο όγκος του πακέτου, το βάρος του, το επείγον ή μη της παράδοσής του και η χιλιομετρική απόσταση μεταξύ των σημείων παραλαβής – παράδοσης , το κόστος αποστολής του κάθε πακέτου μπορεί να υπολογιστεί και φυσικά δεν είναι κάτι πέρα από έναν αριθμό. Είναι φανερό ότι σε αυτό τον τρόπο λειτουργίας, ανεξάρτητα από το πόσοι είναι οι νευρώνες στο επίπεδο εισόδου, στο επίπεδο εξόδου θα έχουμε έναν μόνο νευρώνα ο οποίος θα δίνει το τελικό αποτέλεσμα του δικτύου, το κόστος αποστολής του πακέτου.

Classification: Σε αυτό τον τρόπο λειτουργίας το νευρωνικό δίκτυο καλείται να παράγει όχι έναν αριθμό αλλά μία ταξινόμηση δεδομένων σε κάποιες προκαθορισμένες κλάσεις. Ουσιαστικά περιμένουμε από το δίκτυο να υποδείξει σε ποια κατηγορία συνόλου κατηγοριών, ανήκουν τα δεδομένα που αποτελούν την είσοδο στο δίκτυο. Για παράδειγμα, ας υποθέσουμε ότι έχουμε ένα πλήθος σκύλων το οποίο επιθυμούμε να ταξινομήσουμε σε τρεις κατηγορίες (κλάσεις): Γερμανικούς Ποιμενικούς, Μπουλντόγκ και Μπίγκλ. Παρέχοντας στο νευρωνικό δίκτυο ως είσοδο δεδομένα όπως το ύψος του σκύλου, το βάρος του, το χρώμα του τριχώματος του, την ηλικία του και το φύλλο του μπορούμε να καθορίσουμε σε ποια κλάση ανήκει το ζώο. Παρατηρήστε εδώ ότι ενώ το επίπεδο εισόδου, αποτελείται και πάλι από τόσους κόμβους όσους υποδεικνύει η είσοδος στο νευρωνικό δίκτυο, το επίπεδο εξόδου δεν αποτελείται από έναν κόμβο αλλά από τόσους κόμβους όσες και οι κλάσεις στις οποίες καλείται το νευρωνικό δίκτυο να ταξινομήσει τα δεδομένα. Στο συγκεκριμένο παράδειγμα, το επίπεδο εισόδου θα αποτελείται από πέντε κόμβους (ένας για κάθε παράμετρο της εισόδου – ύψος, βάρος, χρώμα, ηλικία, φύλλο) και το επίπεδο εξόδου θα αποτελείται από 3 κόμβους, έναν για κάθε μία από τις τρεις κλάσεις (Γερμανικός Ποιμενικός, Μπουλντόγκ, Μπίγκλ).

2.4. Δομικά Χαρακτηριστικά Τεχνητού Νευρωνικού Δικτύου.

Αφού περιγράφηκε η δομή, η σύσταση και οι τρόποι λειτουργίας ενός ΤΝΔ, είναι πιο εύκολο τώρα να καταλάβουμε πως λειτουργεί. Βασικές έννοιες για την περιγραφή ενός νευρωνικού δικτύου είναι η είσοδος, η κανονικοποίηση εισόδου (input normalization), η συνάρτηση ενεργοποίησης (activation function), τα συναπτικά βάρη (synaptic weights), ο νευρώνας πόλωσης (bias neuron), και η έξοδος του νευρώνα.

2.4.1 Είσοδος Τεχνητού Νευρωνικού Δικτύου - Κανονικοποίηση.

Τα ΤΝΔ είναι σχεδιασμένα ώστε να δέχονται σαν είσοδο διανύσματα από αριθμούς κινητής υποδιαστολής. Για παράδειγμα ας υποθέσουμε ότι κατασκευάζουμε ένα νευρωνικό δίκτυο του οποίου στόχος είναι να κατανείμει λουλούδια Ίριδας σε ένα από τα τρία είδη του λουλουδιού (setosa, versicolor και virginica) βάση μετρήσεων που παρέχονται στο νευρωνικό δίκτυο ως είσοδος. Οι μετρήσεις που καλείται το να επεξεργαστεί το νευρωνικό δίκτυο, είναι οι μετρήσεις που περιέχονται στο Iris Dataset [7].

“Sepal length”, “Sepal Width”, “Petal length”, “Petal width”, “Species”
5.1, 3.5, 1.4, 0.2, ” setosa”
4.9, 3.0, 1.4, 0.2, “setosa”
4.7, 3.2, 1.3, 0.2, “setosa”
6.3, 3.3, 6.0, 2.5, “virginica”
5.8, 2.7, 5.1, 1.9, “virginica”
7.1, 3.0, 5.9, 2.1, “virginica”
7.0, 3.2, 4.7, 1.4, “versicolor”
6.4, 3.2, 4.5, 1.5, “versicolor”
6.9, 3.1, 4.9, 1.5, “versicolor”

Εικόνα 8. Δείγμα από το Σετ Δεδομένων Iris Dataset

Το συγκεκριμένο σετ δεδομένων (Dataset) περιέχει μετρήσεις λουλουδιών Ίριδας που αφορούν το μήκος του φυλλώματος του λουλουδιού, το πλάτος του φυλλώματος, το μήκος του πετάλου του ανθού, το πλάτος του πετάλου του ανθού και το είδος στο οποίο αντιστοιχεί το

φυτό με τη συγκεκριμένη πλειάδα μετρήσεων. Το Σετ Δεδομένων είναι σε μορφή .CSV αρχείου και έχει τη μορφή που υποδεικνύεται στην Εικόνα 8. Μία πιθανή είσοδος στο νευρωνικό δίκτυο, λοιπόν, μπορεί να είναι η [**7.1, 3.0, 5.9, 2.1**] και η αναμενόμενη έξοδος γι αυτή την είσοδο να είναι η [0,1,0], εάν υποθέσουμε ότι η κλάση virginica έχει ανατεθεί στο δεύτερο κόμβο του επιπέδου εξόδου (θυμηθείτε ότι στην παράγραφο 2.3 είπαμε πως σε λειτουργία Classification, το επίπεδο εξόδου έχει τόσους κόμβους όσες και οι κλάσεις στις οποίες καλείται το νευρωνικό να ταξινομήσει τα δεδομένα).

Μία συνήθης πρακτική για ταχύτερη εκπαίδευση (σύγκλιση νευρωνικού δικτύου) είναι η κανονικοποίηση. Κανονικοποίηση ονομάζεται η μεταφορά των δεδομένων σε ένα συγκεκριμένο εύρος, το οποίο συνήθως είναι από το -1 έως το 1 ή από το 0 έως το 1. Το εύρος στο οποίο θα μεταφερθούν τα δεδομένα συχνά ορίζεται από τις συναρτήσεις ενεργοποίησης που επιλέγονται για το κάθε επίπεδο. Εάν, για παράδειγμα, για το επίπεδο εισόδου επιδεχθεί η υπερβολική εφαπτομένη (hyperbolic tangent: TANH) ως συνάρτηση ενεργοποίησης τότε το εύρος στο οποίο θα μεταφερθούν τα δεδομένα πρέπει να είναι από -1 έως +1. Στο τμήμα του Iris Dataset που παρατίθεται στην Εικόνα 8 βλέπουμε πως το μήκος και πλάτος του πετάλου του ανθού κυμαίνονται από 1.0 έως 6.9 και 0.1 έως 2.5 εκατοστά αντίστοιχα ενώ το μήκος και πλάτος του φυλλώματος του λουλουδιού κυμαίνονται από 7.9 έως 4.3 και 4.4 έως 2.0 εκατοστά αντίστοιχα. Για να κανονικοποιήσουμε τα δεδομένα στο επιθυμητό εύρος (εδώ -1 έως +1) χρησιμοποιούμε τη συνάρτηση:

$$f_{(x)} = \frac{(x - d_L)(n_H - n_L)}{(d_H - d_L)} + n_L$$

Όπου x είναι η τιμή που θέλουμε να κανονικοποιήσουμε, d_H και d_L είναι η μέγιστη και η ελάχιστη τιμή των δεδομένων αντίστοιχα, ενώ n_H και n_L είναι η μέγιστη και η ελάχιστη τιμή του εύρους στο οποίο θέλουμε να μεταφέρουμε την τιμή που κανονικοποιούμε. Επομένως, για το παράδειγμα εισόδου που δώσαμε παραπάνω [**7.1, 3.0, 5.9, 2.1**]:

- 7.1. Τιμή μήκους φυλλώματος λουλουδιού. $f_{(x)} = \frac{(7.1-4.3)(1.0-(-1.0))}{(7.9-4.3)} + (-1.0) = 0.56$
- 3.0. Τιμή πλάτους φυλλώματος λουλουδιού. $f_{(x)} = \frac{(3.0-2.0)(1.0-(-1.0))}{(4.4-2.0)} + (-1.0) = -0.17$
- 5.9. Τιμή μήκους πετάλου ανθού. $f_{(x)} = \frac{(5.9-1.0)(1.0-(-1.0))}{(6.9-1.0)} + (-1.0) = 0.66$

- 2.1. Τιμή πλάτους πετάλου ανθού. $f_{(x)} = \frac{(2.1-0.1)(1.0-(-1.0))}{(2.5-0.1)} + (-1.0) = 0.696$

Άρα πλέον η είσοδος στο νευρωνικό δίκτυο δεν είναι το διάνυσμα [**7.1, 3.0, 5.9, 2.1**] αλλά το διάνυσμα **[0.56, -0.17, 0.66, 0.696]**.

Αντίστοιχα από τη στιγμή που τα δεδομένα εισόδου είναι κανονικοποιημένα, θα είναι κανονικοποιημένα και τα δεδομένα εξόδου τα οποία για να τα αποκανονικοποιήσουμε χρησιμοποιούμε τη συνάρτηση:

$$f_{(x)} = \frac{(d_L - d_H)x - (n_H * d_L) + (d_H * n_L)}{(n_L - n_H)}$$

Επομένως εάν σαν έξοδο του νευρωνικού πάρουμε την κανονικοποιημένη τιμή 0.56, μπορούμε να δούμε σε ποια τιμή αντιστοιχεί στο κανονικό εύρος των μετρήσεων:

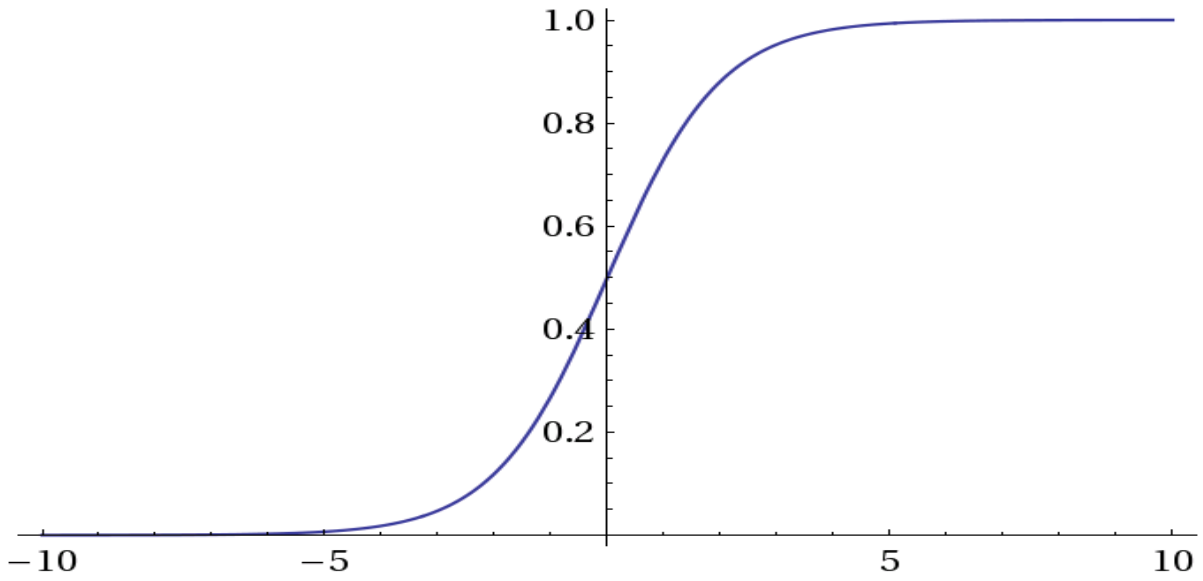
$$f_{(0,56)} = \frac{(4.3-7.9)0.56-(1*4.3)+(7.9*(-1))}{(-1-(1))} = 7.1$$

2.4.2 Συνάρτηση Ενεργοποίησης

Οι περισσότερες μονάδες (κόμβοι - νευρώνες) ενός νευρωνικού δικτύου, διαμορφώνουν την έξοδο τους χρησιμοποιώντας κάποια συνάρτηση, η οποία ονομάζεται συνάρτηση ενεργοποίησης. Η συνάρτηση ενεργοποίησης στη συνέχεια παρέχει μία τιμή η οποία ονομάζεται ενεργοποίηση (activation) της μονάδας (νευρώνα – κόμβου). Συναρτήσεις με φραγμένο εύρος τιμών ονομάζονται περιοριστικές συναρτήσεις, όπως π.χ. η συνάρτηση υπερβολικής εφαπτομένης (εύρος τιμών -1 έως +1) και η σιγμοειδής συνάρτηση (εύρος τιμών από 0 έως +1). Οι συναρτήσεις ενεργοποίησης είναι χρήσιμες γιατί εισάγουν μη γραμμικότητες στο δίκτυο, οι οποίες είναι απαραίτητες για τη μοντελοποίηση περιπλοκότερων προβλημάτων. Ένα παράδειγμα συνάρτησης ενεργοποίησης είναι η σιγμοειδής συνάρτηση, η οποία πρέπει να χρησιμοποιείται μόνο όταν η έξοδος του νευρωνικού δικτύου αναμένεται να είναι πάντα θετική και αυτό επειδή η σιγμοειδής συνάρτηση παράγει μόνο θετική έξοδο. Η συνάρτηση της σιγμοειδούς είναι:

$$S(t) = \frac{1}{1 + e^{-t}}$$

Η συνάρτηση αυτή “μεταφέρει” αρνητικούς αριθμούς στο θετικό επίπεδο (Εικόνα 9).



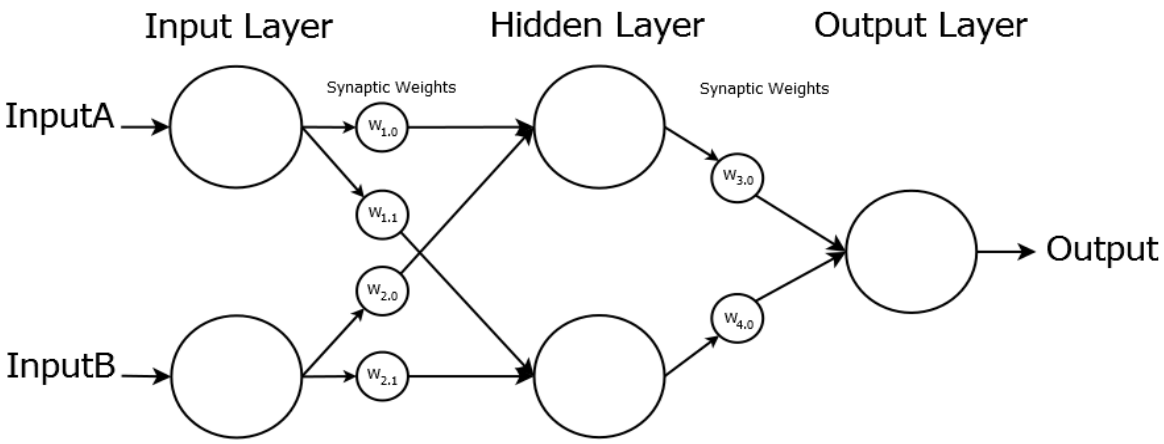
Εικόνα 9. Η γραφική παράσταση τις σιγμοειδούς συνάρτησης.

Αν και η σιγμοειδής συνάρτηση είναι πολύ συχνή επιλογή στο σχεδιασμό νευρωνικών δικτύων, είναι σημαντικό να χρησιμοποιείται μόνο όταν η έξοδος του νευρωνικού δικτύου, αναμένεται να παίρνει μόνο θετικές τιμές. Πρέπει να σημειωθεί εδώ ότι στο επίπεδο εισόδου δεν ανατίθεται συνάρτηση ενεργοποίησης.

2.4.3. Συναπτικά Βάρη

Όπως έχουμε δει μέχρι εδώ, ένα νευρωνικό δίκτυο αποτελείται από ένα σύνολο νευρώνων (κόμβων). Κάθε κόμβος έχει ως είσοδο την έξοδο κάθε κόμβου του προηγούμενου επιπέδου και κάθε κόμβος διαθέτει την έξοδο του ως είσοδο σε κάθε κόμβο του επόμενου επιπέδου. Σε κάθε μία από αυτές τις συνδέσεις αντιστοιχεί ένα βάρος, το οποίο ονομάζεται συναπτικό βάρος. Το βάρος κάθε σύνδεσης, δείχνει πόσο επηρεάζει η σύνδεση αυτή τη συνολική λειτουργία του νευρωνικού δικτύου. Ένα μεγάλο βάρος υποδηλώνει ότι η σύνδεση που έχει το βάρος αυτό, παίζει μεγάλο ρόλο στον υπολογισμό της εξόδου του νευρωνικού δικτύου και ένα μικρό βάρος υποδηλώνει ότι η σύνδεση που έχει το συγκεκριμένο βάρος συμμετέχει λίγο στον υπολογισμό της εξόδου. Έτσι ένας κόμβος Α, διαθέτει σαν είσοδο σε κάποιον κόμβο Β του επόμενου επιπέδου, την έξοδο του (το ποια είναι η έξοδος ενός νευρώνα θα μελετηθεί σε επόμενο κεφάλαιο) πολλαπλασιασμένη επί το βάρος της σύνδεσης μεταξύ του κόμβου Α με τον κόμβο Β και μετασχηματισμένη κατά τη συνάρτηση ενεργοποίησης που έχει επιλεγεί για το επίπεδο στο

οποίο ανήκει ο νευρώνας που διαθέτει την έξοδο του ως είσοδο στο επόμενο επίπεδο (εδώ ο κόμβος Α).

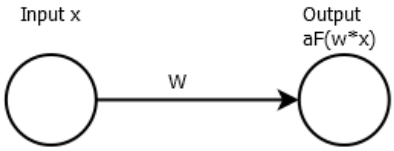


Εικόνα 10: Τα συναπτικά βάρη [$w_{1.0}, w_{1.1}, \dots, w_{4.0}$] των συνδέσεων του νευρωνικού δικτύου.

Αρχικά, όταν το δίκτυο σχεδιάζεται, τα βάρη αυτά έχουν τυχαίες τιμές και για τη σωστή ρύθμιση τους απαιτείται εκπαίδευση του δικτύου με γνωστά δεδομένα. Κατά την εκπαίδευση, τα συναπτικά βάρη αναπροσαρμόζονται έτσι ώστε για δεδομένη είσοδο το νευρωνικό δίκτυο να παρέχει έξοδο ίση στο ποσοστό που ορίζεται, με την αναμενόμενη.

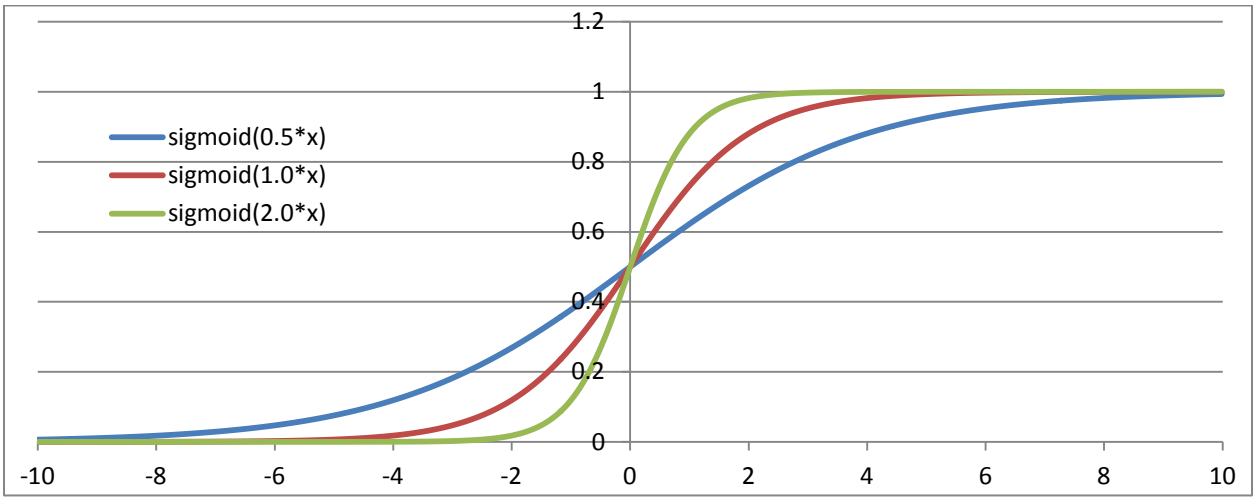
2.4.4. Νευρώνας Πόλωσης (Bias Neuron)

Ο νευρώνας πόλωσης είναι ένας νευρώνας, ρυθμισμένος να έχει πάντα μη μηδενική έξοδο (ίση με 1). Ο νευρώνας αυτός δεν αποτελεί απαραίτητο στοιχείο ενός νευρωνικού δικτύου επειδή υπάρχουν προβλήματα που λύνονται ακόμα και χωρίς τη χρήση του, όμως ακόμα και στα προβλήματα στα οποία δεν είναι απαραίτητος, η χρήση του επιτρέπει βελτιστοποίηση του δικτύου. Κάθε επίπεδο του δικτύου, εκτός από το επίπεδο εξόδου, μπορεί να έχει έναν νευρώνα πόλωσης, ο οποίος συνδέεται με όλους τους κόμβους του επόμενου επιπέδου. Οι συνδέσεις του νευρώνα πόλωσης με τους κόμβους του επόμενου επιπέδου έχουν τα δικά τους συναπτικά βάρη, Υποθέστε, για παράδειγμα, ένα απλό νευρωνικό δίκτυο που αποτελείται από ένα κόμβο εισόδου και ένα κόμβο εξόδου όπως αυτό της Εικόνας 11 [8].



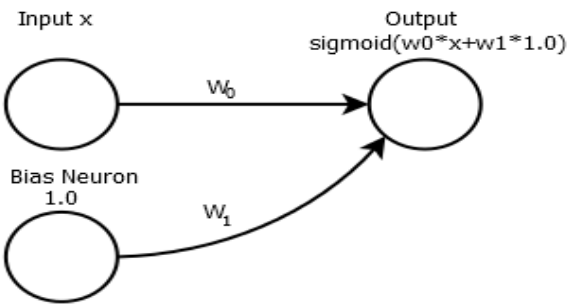
Εικόνα 11: Ένα απλό ΤΝΔ, δυο επιπέδων.

Η έξοδος του δικτύου υπολογίζεται πολλαπλασιάζοντας την είσοδο x επί το βάρος w και μετασχηματίζοντας το αποτέλεσμα με μία συνάρτηση ενεργοποίησης (εν προκειμένω τη σιγμοειδή συνάρτηση). Στην Εικόνα 12 φαίνεται η έξοδος του δικτύου για διάφορες τιμές του w.



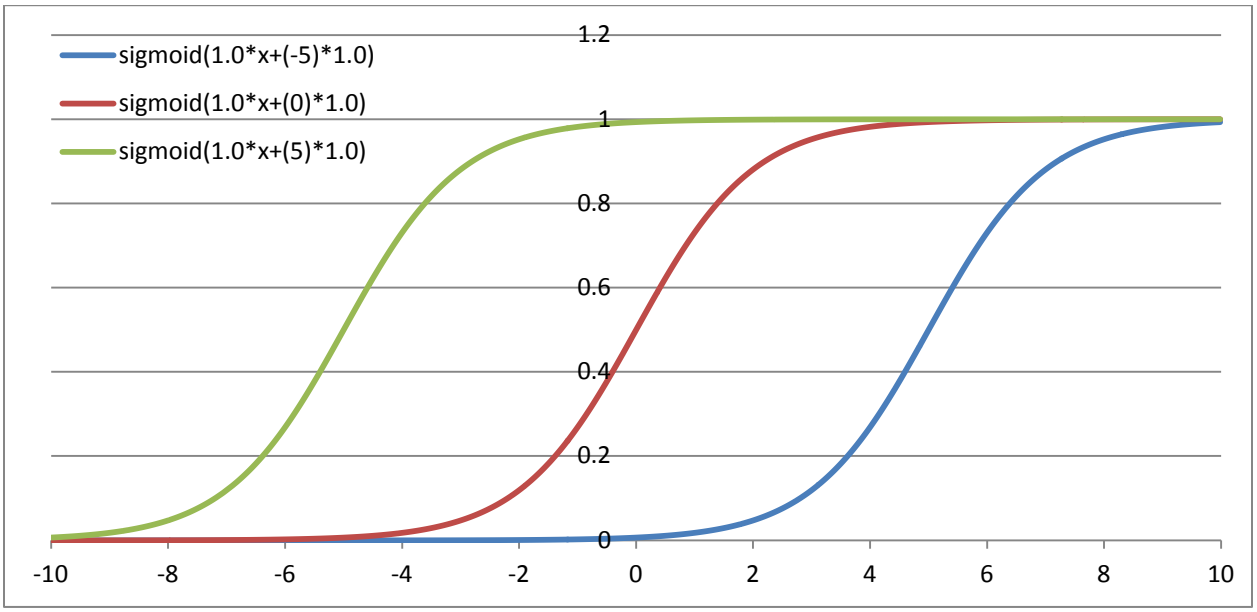
Εικόνα 12: Η έξοδος του νευρωνικού δικτύου για βάρη w ίσα με 0.5, 1 και 2.

Αλλάζοντας το βάρος της σύνδεσης μπορούμε να αλλάξουμε την κλίση της εξόδου, όμως δεν μπορούμε να μετατοπίσουμε την καμπύλη εξόδου προς τα δεξιά, που σημαίνει ότι για θετικές τιμές του x το δίκτυο δεν μπορεί να δώσει έξοδο ίση με μηδέν. Γι αυτό το λόγο χρησιμοποιούμε το νευρώνα πόλωσης ο οποίος όπως είπαμε είναι ρυθμισμένος να έχει έξοδο πάντα ίση με 1.



Εικόνα 13. Απλό ΤΝΔ δυο επιπέδων με νευρώνα πόλωσης.

Πλέον η έξοδος του κόμβου εξόδου δεν διαμορφώνεται μόνο από το μετασχηματισμό της εισόδου επί το βάρος σύνδεσης [$sigmoid(w_0 * x)$], αλλά από το μετασχηματισμό του αθροίσματος γινομένων της εισόδου επί το βάρος σύνδεσης w_0 συν την έξοδο του κόμβου πόλωσης επί το βάρος σύνδεσης w_1 [$sigmoid(w_0 * x + 1.0 * w_1)$].



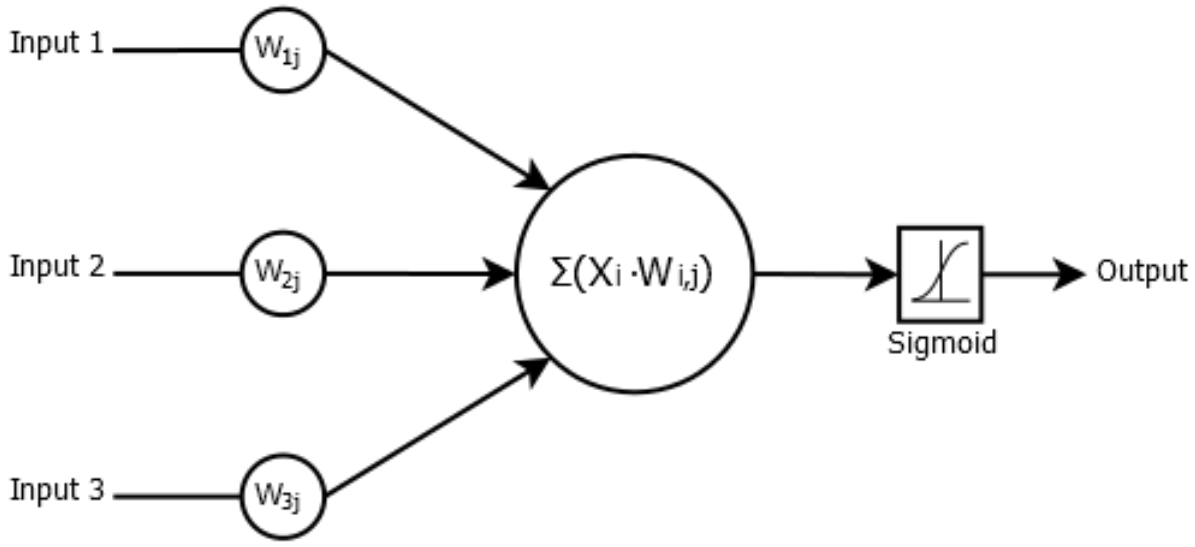
Εικόνα 14: Η μετατοπισμένη έξοδος του δικτύου.

Όπως βλέπουμε και στην Εικόνα 14 για διαφορετικά συναπτικά βάρη ανάμεσα στον κόμβο πόλωσης και τον κόμβο εξόδου, η έξοδος του δικτύου μετατοπίζεται στον οριζόντιο άξονα. Συγκεκριμένα για βάρος σύνδεσης ίσο με -5 βλέπουμε πως με είσοδο 2 μπορούμε να πάρουμε έξοδο πολύ κοντά στο 0. Πρέπει να σημειωθεί εδώ ότι νευρώνας πόλωσης δεν χρησιμοποιείται στο επίπεδο εξόδου.

2.4.5. Έξοδος Νευρώνα

Μέχρι τώρα είδαμε ποιοι είναι οι μηχανισμοί που χρησιμοποιούνται ώστε να λειτουργήσει ένα νευρωνικό δίκτυο και να διαθέσει έξοδο στον έξω κόσμο. Η έξοδος του νευρωνικού δικτύου είναι η έξοδος του/των νευρώνα/νευρωνων του επιπέδου εξόδου. Επομένως εάν καταλάβουμε ποια είναι η έξοδος ενός νευρώνα του δικτύου μπορούμε, συνεπαγωγικά, να κατανοήσουμε ποια είναι η έξοδος ολόκληρου του δικτύου. Σύμφωνα με ότι έχουμε πει μέχρι τώρα ένας νευρώνας έχει ως είσοδο τη μετασχηματισμένη, βάσει της συνάρτησης ενεργοποίησης, έξοδο ενός νευρώνα προηγούμενου επιπέδου. Είδαμε όμως ότι η είσοδος ενός οποιουδήποτε νευρώνα είναι δυνατόν να προέρχεται τόσο από ένα μόνο νευρώνα του προηγούμενου επιπέδου όσο και από ένα σύνολο κατά μέγιστο n κόμβων, όπου n το πλήθος των κόμβων του προηγούμενου επιπέδου. Δεδομένου αυτού μπορούμε να πούμε ότι η έξοδος κάθε κόμβου (για παράδειγμα του jοστού κόμβου) περιγράφεται από δυο συναρτήσεις:

1. $U_j = \sum(x_i \times W_{ij})$ Συνάρτηση[1]
2. $Y_j = F_{th}(U_j)$ Συνάρτηση[2]



Εικόνα 15: Οι εσωτερικές λειτουργίες ενός νευρώνα και η συνάρτηση ενεργοποίησης.

Όπως φαίνεται και στην Εικόνα 15 για κάθε νευρώνα j, σε οποιοδήποτε επίπεδο, κάθε είσοδος X_i, από τις i εισόδους σε αυτόν το νευρώνα, πολλαπλασιάζεται με ένα βάρος W_{ij}. Τα γινόμενα

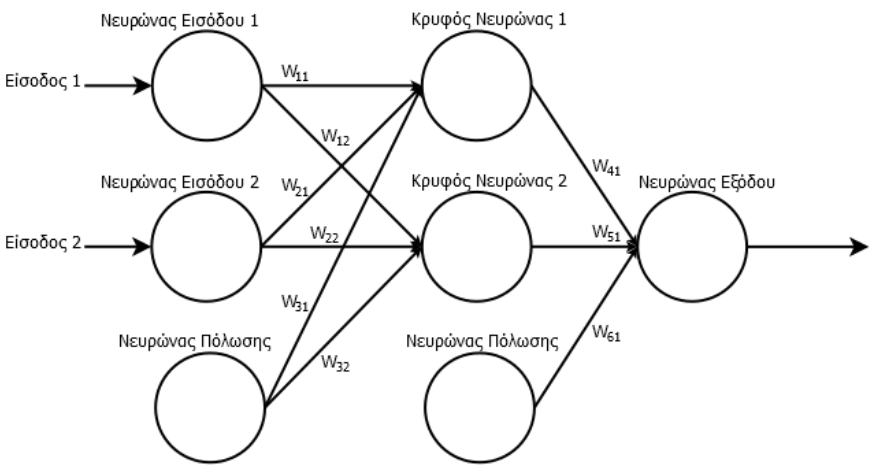
προστίθενται και σχηματίζουν την εσωτερική ποσότητα U_j του νευρώνα. Στη συνέχεια αυτή η ποσότητα μετασχηματίζεται βάσει της συνάρτησης ενεργοποίησης και σχηματίζει την ποσότητα Y_j. Η ποσότητα αυτή είναι η έξοδος του νευρώνα και αποτελεί είσοδο σε νευρώνες του επόμενου επιπέδου ή έξοδο του νευρωνικού δικτύου εάν ο νευρώνας υπό μελέτη, ανήκει στο επίπεδο εξόδου.

Παράδειγμα κατανόησης λειτουργίας Νευρωνικού δικτύου.

Προκειμένου να γίνει πλήρως κατανοητός ο τρόπος με τον οποίο λειτουργεί ένα νευρωνικό δίκτυο υποθέστε ένα απλό Feed Forward δίκτυο το οποίο υλοποιεί τη λογική πράξη XOR με δυο εισόδους [9]. Σύμφωνα με όσα έχουμε πει μέχρι τώρα ένα τέτοιο νευρωνικό δίκτυο :

- Θα έχει δυο νευρώνες στο επίπεδο εισόδου - έναν για κάθε κανάλι εισόδου του αντίστοιχου λογικού τελεστή XOR.
- Μπορεί να έχει ή να μην έχει κρυφό επίπεδο. Σύμφωνα με τους εμπειρικούς κανόνες που αναφέραμε στη παράγραφο 2.2 Αρχιτεκτονική Νευρωνικού Δικτύου, η προσθήκη κρυφού επιπέδου, ευνοεί την αποδοτικότητα του δικτύου.
- Εάν έχει κρυφό επίπεδο, τότε βάσει εμπειρικών κανόνων, το πλήθος νευρώνων του επιπέδου αυτού θα είναι ίσο με το μέσο όρο των νευρώνων που αποτελούν τα υπόλοιπα επίπεδα ή ίσο με το διπλάσιο των κόμβων του επιπέδου εισόδου. Σε κάθε περίπτωση όμως ο ιδανικός αριθμός κρυφών κόμβων βρίσκεται με δοκιμές και μετρήσεις αποτελεσμάτων.
- Το επίπεδο εξόδου θα έχει ένα νευρώνα δεδομένου ότι η επιθυμητή λειτουργία του δικτύου είναι Regression (ως έξοδος δικτύου αναμένεται ένας αριθμός, 0 ή 1)
- Δεδομένου ότι υλοποιούμε το λογικό τελεστή XOR, επομένως η έξοδος που περιμένουμε – καθώς και τα δεδομένα που θα χρησιμοποιήσουμε, παίρνουν μόνο θετικές τιμές, ως συνάρτηση ενεργοποίησης θα χρησιμοποιήσουμε τη σιγμοειδή συνάρτηση.

Στην Εικόνα 16, φαίνεται το νευρωνικό δίκτυο που προκύπτει βάσει των κανόνων που μόλις περιγράψαμε. Υποθέτουμε ότι τα βάρη στις συνδέσεις των νευρώνων είναι γνωστά και είναι αυτά που υποδεικνύονται στον πίνακα της Εικόνας 16.



W#	Τιμή	Σύνδεση
W_{11}	5	ΕΙΣ 1 -> ΚΝ1
W_{12}	-6	ΕΙΣ 1 -> ΚΝ2
W_{21}	-6	ΕΙΣ 2 -> ΚΝ1
W_{22}	6	ΕΙΣ 2 -> ΚΝ2
W_{31}	-3	ΝΠ -> ΚΝ1
W_{32}	-3	ΝΠ -> ΚΝ2
W_{41}	10	ΚΝ 1 -> ΝΕ
W_{51}	10	ΚΝ 2 -> ΝΕ
W_{61}	-5	ΝΠ -> ΝΕ

Εικόνα 16. Ένα απλό Feed Forward νευρωνικό δίκτυο που υλοποιεί το λογικό τελεστή XOR.

Υποθέστε τώρα ότι η είσοδος που δέχεται το νευρωνικό δίκτυο, είναι ίση με 0 για την Είσοδο 1 και 1 για την Είσοδο 2. Ο νευρώνας πόλωσης έχει επίσης έξοδο ίση με 1. Για να δούμε ποια θα είναι η έξοδος του νευρώνα εξόδου πρέπει να δούμε ποια θα είναι η είσοδος του. Η είσοδος του είναι ίση με το άθροισμα των εξόδων των κόμβων του κρυφού επιπέδου. Για κάθε ένα από τους κόμβους του κρυφού επιπέδου η είσοδος του, είναι το άθροισμα των εξόδων των κόμβων του επιπέδου εισόδου. Θυμηθείτε ότι στο επίπεδο εισόδου δεν ανατίθεται συνάρτηση ενεργοποίησης και ότι για κάθε κόμβο σε κάθε επίπεδο του δικτύου ισχύουν οι συναρτήσεις [1] και [2]. Άρα :
Για τον Κρυφό Νευρώνα 1:

- Είσοδος είναι το άθροισμα των εξόδων των νευρώνων του προηγούμενου επιπέδου. Άρα: $U_j = \sum(x_i \times W_{ij}) \Rightarrow U_j = (W_{11} * NE1) + (W_{21} * NE2) + (W_{31} * N\Pi)$, επομένως $U_j = (5 * 0) + (-6 * 1) + (-3 * 1) = -9$
- Έξοδος είναι η μετασχηματισμένη βάσει συνάρτησης ενεργοποίησης είσοδος του νευρώνα: $Y_j = F_{th}(U_j) \Rightarrow Y_j = \frac{1}{1+e^{-(-1)}} = 0.000123$ (για F_{th} ιση με τη σιγμοειδή συνάρτηση)

Για τον Κρυφό Νευρώνα 2:

- Είσοδος: $U_j = \sum(x_i \times W_{ij}) \Rightarrow U_j = (W_{12} * NE1) + (W_{22} * NE2) + (W_{32} * N\Pi)$
  Άρα: $U_j = (-6 * 0) + (6 * 1) + (-3 * 1) = 3$
- Έξοδος: $Y_j = F_{th}(U_j) \Rightarrow Y_j = \frac{1}{1+e^{-(3)}} = 0.95257$

Επομένως η έξοδος του κρυφού νευρώνα 1 είναι ίση με [0.000123] και η έξοδος του κρυφού νευρώνα 2 είναι ίση με [0.95257]. Οι δυο αυτοί δεκαδικοί αριθμοί αποτελούν την είσοδο του νευρώνα εξόδου. Επομένως

Για το νευρώνα Εξόδου:

- Είσοδος: $U_j = \sum(x_i \times W_{ij}) \Rightarrow U_j = (W_{41} * KN1) + (W_{51} * KN2) + (W_{61} * N\Pi)$
  Άρα: $U_j = (10 * 0.000123) + (10 * 0.95257) + (-5 * 1) = 4.52693$
- Έξοδος: $Y_j = F_{th}(U_j) \Rightarrow Y_j = \frac{1}{1+e^{-(4,52693)}} = 0.989301$

Βλέπουμε πως το νευρωνικό δίκτυο για είσοδο ίση με [0,1] δίνει έξοδο ίση με [0.99], μία τιμή πολύ κοντά στη μονάδα που είναι και η αναμενόμενη τιμή δεδομένου ότι το νευρωνικό δίκτυο μοντελοποιεί το λογικό τελεστή XOR.

3. Περιγραφή Εφαρμογής

3.1. Στόχος της Εργασίας

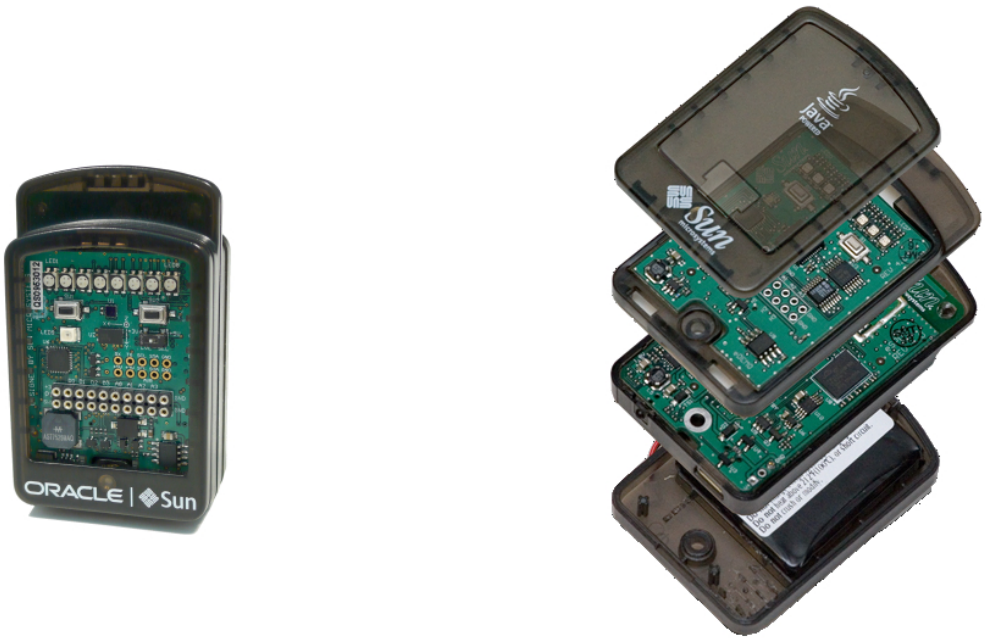
Στόχος της παρούσας εργασίας είναι η δημιουργία ενός έξυπνου συστήματος εντοπισμού πυρκαγιάς το οποίο βασίζεται σε ασύρματα δίκτυα αισθητήρων και νευρωνικά δίκτυα. Τα περισσότερα από τα ήδη υπάρχοντα συστήματα εντοπισμού πυρκαγιάς χρησιμοποιούν αισθητήρες μέτρησης αέριων εκπομπών, αισθητήρες έντασης άνεμου, υγρασίας και θερμοκρασίας προκειμένου να συλλέξουν πληροφορία για την κατάσταση του περιβάλλοντα χώρου. Οι μετρήσεις που λαμβάνονται από τους αισθητήρες αυτούς υπόκεινται σε επεξεργασία έτσι ώστε να εξαχθεί συμπέρασμα για την ύπαρξη ή μη φωτιάς. Οι συναγερμοί που δίνονται από αυτά τα συστήματα βασίζονται σε τιμές κατωφλιού για μέγιστες ή ελάχιστες τιμές στα μεγέθη θερμοκρασίας η υγρασίας αντίστοιχα, σε αισθητήρες καπνού και σε αισθητήρες εκπομπών αερίων. Το μειονέκτημα σε αυτά τα συστήματα είναι ότι αφενός οι τιμές κατωφλίου είναι πολύ δύσκολο να οριστούν ικανοποιητικά (για παράδειγμα εάν για τη θερμοκρασία έχει οριστεί ως τιμή κατωφλίου αυτή των 30^{o} C (από τους 30 βαθμούς και πάνω, υπάρχει πυρκαγιά), τους χειμερινούς μήνες μπορεί το σύστημα εντοπισμού να λειτουργεί ικανοποιητικά αλλά κατά τους καλοκαιρινούς μήνες θα δώσει πολλούς εσφαλμένους συναγερμούς) και αφετέρου ότι υπάρχουν πολλοί λόγοι για τους οποίους ένας αισθητήρας μπορεί να λαμβάνει λανθασμένες μετρήσεις – για παράδειγμα ένας αισθητήρας καπνού μπορεί να «ξεγελαστεί» από μία μικρή ελεγχόμενη φωτιά ή από τον καπνό ενός τσιγάρου. Στην παρούσα εργασία μελετάται εάν ένα έξυπνο σύστημα, που αποτελείται από δίκτυο αισθητήρων και νευρωνικό δίκτυο, μπορεί να παρέχει έγκυρα και έγκαιρα αποτελέσματα για την ύπαρξη ή μη πυρκαγιάς με ποσοστό σφάλματος μικρότερο από αυτό των παραδοσιακών συστημάτων.

3.2. Εργαλεία που χρησιμοποιήθηκαν

3.2.1. Η πλατφόρμα ανάπτυξης

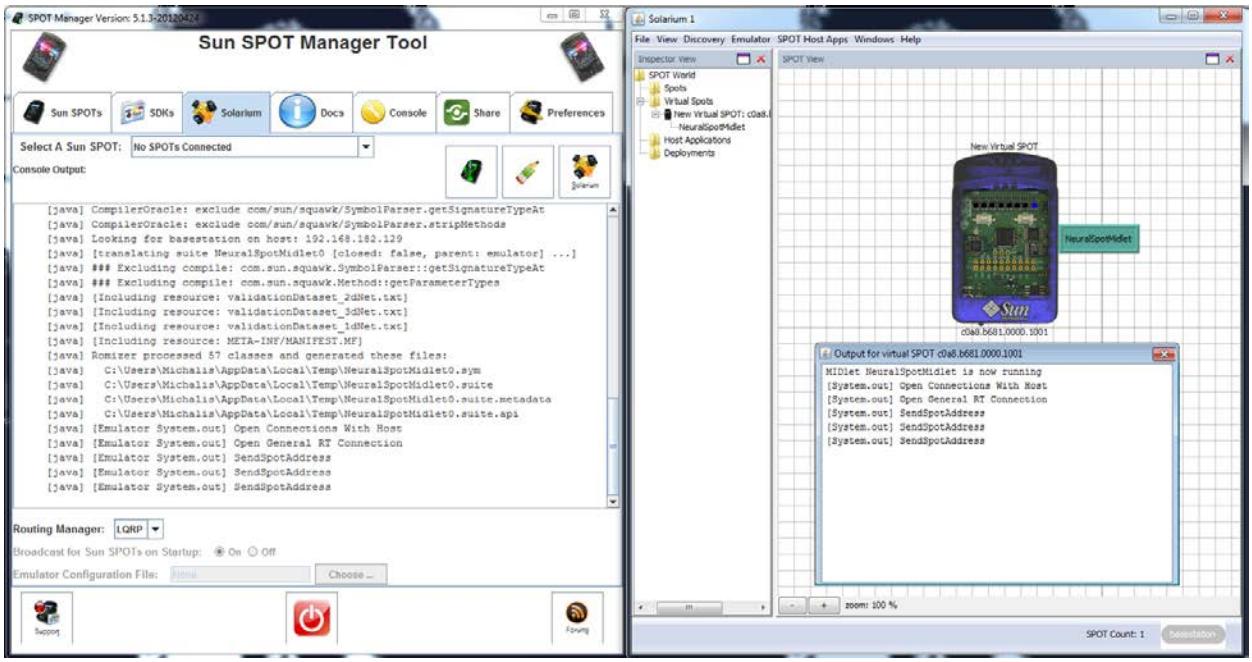
Το σύστημα υλοποιήθηκε στην πλατφόρμα της Oracle, SunSPOT, μία μικρή ασύρματη πλατφόρμα τροφοδοτούμενη από μπαταρία. Αρχιτεκτονικά αποτελείται από 3 φυσικά στοιβαγμένα επίπεδα.

- Το πρώτο επίπεδο αποτελείται από τη μπαταρία της πλατφόρμας,
- Το δεύτερο επίπεδο, το επίπεδο επεξεργαστή, αποτελείται από ένα μικροελεγκτή αρχιτεκτονικής 32bit [ARM920T], χρονισμένο στα 180 MHZ, 512Kb μνήμης Ram, 4 MB μνήμης flash, το module ασύρματης επικοινωνίας [Radio 802.15.4] και το module USB διασύνδεσης.
- Το τρίτο επίπεδο, το επίπεδο αισθητήρων, αποτελείται από ένα επιταχυνσιόμετρο τριών αξόνων, έναν αισθητήρα φωτεινότητας, έναν αισθητήρα θερμοκρασίας, ένα πληκτρολόγιο, οκτώ LEDs φάσματος RGB, έξι αναλογικές εισόδους, πέντε εισόδους γενικής χρήσης I/O και τέσσερις ακίδες εξόδου.



Εικόνα 17: Αριστερά η πλατφόρμα SunSPOT. Δεξιά τα επίπεδα της πλατφόρμας.

Η πλατφόρμα SunSPOT εμπεριέχει το Squawk VM, ένα virtual machine για Java Micro edition [j2me ή JavaME], γραμμένο εξολοκλήρου σε Java και υποστηρίζει CLDC 1.1 και MIDP 1.0 (και τα δυο είναι σύνολα βιβλιοθηκών και specification που υποστηρίζονται στην j2me). Η Oracle, διαθέτει δωρεάν το περιβάλλον ελέγχου των SunSPOT, Sun SPOTManager, το οποίο περιέχει και τον εξομοιωτή Solarium, ο οποίος επιτρέπει τη χρήση εικονικών SunSPOT, όμως δεν διατίθεται κάποιο αποκλειστικό περιβάλλον ανάπτυξης εφαρμογών γι αυτά. Αντίθετα η ανάπτυξη εφαρμογών για τα SunSPOT είναι δυνατή στα 2 δημοφιλέστερα IDE για ανάπτυξη εφαρμογών, το Netbeans και το Eclipse καθώς υπάρχουν plug-in και για τα δυο IDE που επιτρέπουν την ανάπτυξη εφαρμογών για SunSPOT σε αυτά.



Εικόνα 18: Το πρόγραμμα ελέγχου Sun SPOTManager και το πρόγραμμα προσομοίωσης Solarium.

Το συνοδευτικό πρόγραμμα ελέγχου Sun SPOTManager υποστηρίζεται στα περισσότερα δημοφιλή λειτουργικά συστήματα των 32bit, όπως MacOS, Linux, Windows. Για τους σκοπούς αυτής της εργασίας, χρησιμοποιήθηκε virtual machine με λειτουργικό σύστημα Windows 7 32bit. Επιπροσθέτως πρέπει να σημειωθεί ότι υπάρχουν δυο κατηγορίες SPOT, τα basestation και τα free range. Είναι διαφορετικά μεταξύ τους τόσο στην κατασκευή όσο και στη λειτουργία. Κατασκευαστικά τα free range SPOT έχουν επαναφορτιζόμενη μπαταρία, εκτελούν .midlet εφαρμογές και έχουν ενδεικτικά led. Αντίθετα τα basestation SPOT δεν έχουν led ή μπαταρία (επαναφορτιζόμενη ή μη) και για να λειτουργήσουν πρέπει να είναι συνδεδεμένα σε κάποιο υπολογιστή (μέσω θύρας USB) ενώ δεν εκτελούν .midlet εφαρμογές. Τα basestation SPOT

αποτελούν μέσο επικοινωνίας ανάμεσα στα free range SPOT και κάποιον υπολογιστή που εκτελεί μία host εφαρμογή. Μία host εφαρμογή είναι μία Java εφαρμογή η οποία εκτελείται σε κάποιον υπολογιστή και έχει πρόσβαση σε ένα υποσύνολο των βιβλιοθηκών (ουσιαστικά υποσύνολο του API) που χρησιμοποιούνται από τα SPOT. Αυτό σημαίνει ότι μία host εφαρμογή μπορεί να επικοινωνήσει με κάποιο free range SPOT με κώδικα, ίδιο με αυτόν που θα χρησιμοποιούσαμε για την επικοινωνία ανάμεσα σε δυο free range SPOT. Σε αυτό το σημείο πρέπει να σημειωθεί ότι η host εφαρμογή δεν εκτελείται στο basestation, αλλά χειρίζεται το basestation μέσω σύνδεσης USB.

3.2.2 Νευρωνικό Δίκτυο – Encog

Από τις διαθέσιμες open source βιβλιοθήκες νευρωνικών δικτύων, που υπάρχουν [DeepDream, Joone, Neuroph, OpenNN], επιλέχτηκε η Encog [10]. Η Encog είναι μια βιβλιοθήκη που υλοποιεί νευρωνικά δίκτυα, η οποία αναπτύχτηκε από την Heaton Research Inc and contributors, είναι ανοιχτού κώδικα και υποστηρίζει μερικές από τις πιο διαδεδομένες γλώσσες προγραμματισμού όπως Java, C++, C και .NET. Περιέχει κλάσεις που μπορούν να δημιουργήσουν μία πληθώρα τύπων νευρωνικών δικτύων, καθώς και κλάσεις για κανονικοποίηση και επεξεργασία δεδομένων για το νευρωνικό δίκτυο. Επιλέχτηκε καθώς είναι το μοναδικό για το οποίο αυτή τη στιγμή υπάρχει port στη JavaME. Αν και το Encog αυτό καθεαυτό είναι ένα πολύ δυνατό εργαλείο, οι δυνατότητες του που μπορούμε να χρησιμοποιήσουμε, είναι περιορισμένες και αυτό γιατί ουσιαστικά, στο δίκτυο αισθητήρων δεν υλοποιείται νευρωνικό δίκτυο με το Encog αλλά με το port του σε j2me, EncogME.
Μερικές κλάσεις [11] του Encog που χρησιμοποιούνται στην παρούσα εργασία είναι, ενδεικτικά, οι εξής:

- **BasicNetwork**: Είναι η κλάση που υλοποιεί ένα νευρωνικό δίκτυο. Επίπεδο, νευρώνες και συνάρτηση ενεργοποίησης δεν αποτελούν μέλη του δικτύου, αλλά είναι ξεχωριστές κλάσεις τις οποίες χρησιμοποιεί η κλάση BasicNetwork (σχέση has-a).
- **BasicLayer**: Είναι η κλάση που υλοποιεί ένα επίπεδο νευρωνικού δικτύου. Το κάθε επίπεδο που δημιουργείται προστίθεται στο υπάρχον νευρωνικό δίκτυο με τη μέθοδο .addLayer(); της κλάσης BasicNetwork. Ο νευρώνας είναι δομικό μέλος της κλάσης (ορίζεται σε αυτήν) ενώ η συνάρτηση μεταφοράς είναι ξεχωριστή κλάση την οποία χρησιμοποιεί η κλάση BasicLayer (σχέση has-a).

- **ActivationSigmoid**: Είναι η κλάση που υλοποιεί τη σιγμοειδή συνάρτηση ενεργοποίησης.
- **MLDataSet**: Το Encog είναι σχεδιασμένο να δέχεται είσοδο που ανήκει σε αυτή την κλάση. Με αλλά λόγια ένα διάνυσμα εισόδου δεν μπορεί να είναι τύπου double[] αλλά πρέπει να είναι τύπου MLDataSet. Η κλάση περιέχει αρκετές μεθόδους εισαγωγής δεδομένων όπως για παράδειγμα την .setData(int index,double data).

Το αντίστοιχο κομμάτι κώδικα που υλοποιεί ένα απλό FeedForward νευρωνικό δίκτυο τριών επιπέδων, με συνάρτηση ενεργοποίησης τη σιγμοειδή συνάρτηση, με 2,4,1 νευρώνες στα επίπεδα εισόδου, κρυφό, εξόδου αντίστοιχα και με νευρώνα πόλωσης (true/false τιμές στα ορίσματα του BasicLayer) στα επίπεδα εισόδου και κρυφό. Αξίζει να σημειωθεί και πάλι ότι το επίπεδο εισόδου δεν έχει συνάρτηση ενεργοποίησης και το επίπεδο εξόδου δεν έχει νευρώνα πόλωσης.

```
this.network = new BasicNetwork();
network.addLayer(new BasicLayer(null,true,2));
network.addLayer(new BasicLayer(new ActivationSigmoid(),true,4));
network.addLayer(new BasicLayer(new ActivationSigmoid(),false,1));
network.getStructure().finalizeStructure();
network.reset();
```

Εικόνα 19: Κώδικας που υλοποιεί ένα απλό FeedForward Δίκτυο 3ων επιπέδων.

3.2.3. Νευρωνικό Δίκτυο EncogME

Το EncogME [12] είναι ένα port του Encog σε j2me. Τη μετάβαση από JavaSE σε JavaME έκανε ο ερευνητής του πανεπιστημίου της S.Florida, κ. Sean Barbeau. Αποτελείται από ένα πολύ μικρό κομμάτι του αρχικού Encog, είναι αρκετό όμως για τη δημιουργία και εκπαίδευση ενός νευρωνικού δικτύου. Για την υλοποίηση της παρούσας εργασίας χρειάστηκε να δημιουργηθούν μερικές ακόμα κλάσεις του αρχικού Encog στο EncogME, όπως οι κλάσεις οι οποίες είναι υπεύθυνες για την κανονικοποίηση δεδομένων (NormalizeAction.java, NormalizedField.java) και η κλάση συνάρτησης ενεργοποίησης υπερβολικής εφαπτομένης(ActivationTANH).

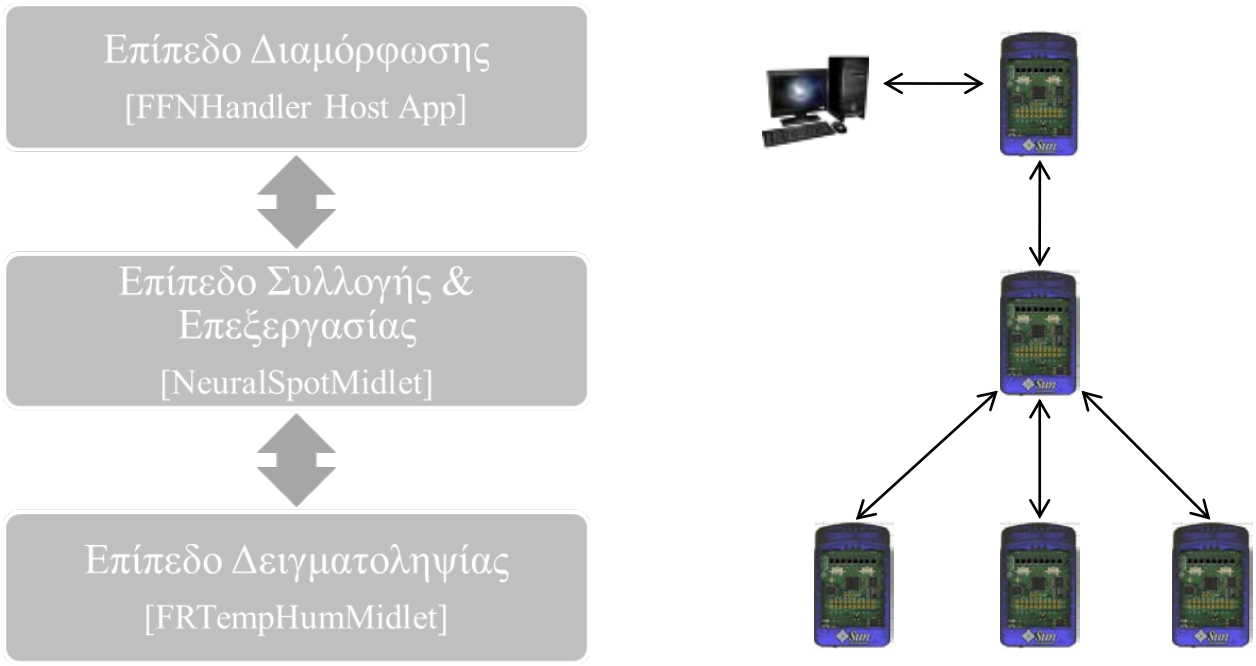
3.3 Αρχιτεκτονική και Λειτουργία Συστήματος.

Για την υλοποίηση του συστήματος, αρχικά θεωρήθηκε ότι υπάρχουν δυο περιοχές μετρήσεων. Οι μετρήσεις που υποδεικνύουν πυρκαγιά και οι μετρήσεις που δεν υποδεικνύουν. Οι μετρήσεις που σηματοδοτούν πυρκαγιά κυμαίνονται στο εύρος [10 - 36] για την υγρασία και [30 - 50 °C] για τη θερμοκρασία. Οι μετρήσεις που δεν σηματοδοτούν φωτιά κυμαίνονται στο εύρος [36 - 100] για την υγρασία και [5 - 29.99] για τη θερμοκρασία. Επιπροσθέτως θεωρήθηκε ότι μετρήσεις υγρασίας στην περιοχή [10 - 36] δεν σηματοδοτούν πυρκαγιά εάν η θερμοκρασία κυμαίνεται στο εύρος[5 - 29.99].

Αρχιτεκτονικά, θεωρήθηκαν τρία επίπεδα ιεραρχίας, με διαφορετικές αρμοδιότητες το καθένα. Καθώς η εκπαίδευση νευρωνικού δικτύου είναι μία διαδικασία που απαιτεί μεγάλη επεξεργαστική ισχύ (και κατ' επέκταση χρόνο), επιλέχτηκε η διαδικασία αυτή, να εκτελείται στον υπολογιστή και όχι σε κάποιο SPOT. Η πλατφόρμα ανάπτυξης λογισμικού SPOTManager δίνει τη δυνατότητα επικοινωνίας ενός υπολογιστή με ελεύθερα SPOT εφόσον συνδεθεί στον υπολογιστή ένα basestation SPOT. Έτσι προκύπτει ένα επίπεδο διαμόρφωσης, στο οποίο δημιουργείται και εκπαιδεύεται ένα νευρωνικό δίκτυο σε υπολογιστή. Αφού εξασφαλιστεί η δημιουργία και εκπαίδευση του νευρωνικού δικτύου, πρέπει να συλλεχθούν μετρήσεις για τα μεγέθη της θερμοκρασίας και υγρασίας του περιβάλλοντα χώρου. Έτσι προκύπτει το επόμενο επίπεδο, το επίπεδο δειγματοληψίας, στο οποίο SPOT τοποθετημένα στο χώρο πραγματοποιούν μετρήσεις και τις διαθέτουν για επεξεργασία. Τα SPOT που λειτουργούν σε αυτό το επίπεδο αναφέρονται ως Samplers. Αν και η επεξεργασία της πληροφορίας θα μπορούσε να γίνει κατευθείαν από το επίπεδο διαμόρφωσης στο οποίο ήδη υπάρχει εκπαιδευμένο νευρωνικό δίκτυο, θυμηθείτε ότι το εν λόγω επίπεδο υλοποιείται σε ηλεκτρονικό υπολογιστή και η φύση της εφαρμογής (ανίχνευση πυρκαγιάς σε δασική έκταση) δεν επιτρέπει την παρατεταμένη χρήση υπολογιστή. Έτσι προκύπτει η ανάγκη δημιουργίας του επιπέδου Συλλογής και Επεξεργασίας το οποίο υλοποιεί εκ νέου ένα νευρωνικό δίκτυο κατάλληλα διαμορφωμένο ώστε να επεξεργαστεί την πληροφορία που είναι διαθέσιμη. Το νευρωνικό δίκτυο που υλοποιείται σε αυτό το επίπεδο είναι ίδιο με το νευρωνικό δίκτυο που δημιουργήθηκε στο επίπεδο διαμόρφωσης. Το SPOT σε αυτό το επίπεδο αναφέρεται ως Aggregator. Σε αυτό το σημείο πρέπει να σημειωθεί ότι δυο νευρωνικά δίκτυα λειτουργούν όμοια και έχουν κοινή έξοδο, δεδομένης της κοινής τους εισόδου, όταν έχουν :

- Ίδιο αριθμό επιπέδων.
- Ίδιο αριθμό νευρώνων σε κάθε επίπεδο.
- Ίδιο αριθμό νευρώνων πόλωσης, στα αντίστοιχα επίπεδα.
- Ίδιες συναρτήσεις ενεργοποίησης.
- Ίδια συναπτικά βάρη.

Επομένως το νευρωνικό δίκτυο του επιπέδου επεξεργασίας, δεν χρειάζεται να εκπαιδευτεί, αλλά παίρνει τα βάρη του, από το ήδη εκπαιδευμένο δίκτυο του επιπέδου διαμόρφωσης, δεδομένου ότι, με βάση τα ανωτέρω, λειτουργεί όμοια με το δίκτυο του επιπέδου διαμόρφωσης. Επίσης δεν υπάρχει επικοινωνία ανάμεσα σε κάποιο Sampler και την host εφαρμογή. Υπάρχει επικοινωνία μόνο ανάμεσα στην host εφαρμογή και τον Aggregator και ανάμεσα στον Aggregator και τα Sampler SPOT. Στην Εικόνα 20 φαίνεται η ιεραρχία στα επίπεδα της εφαρμογής.



Εικόνα 20: Τα επίπεδα της εφαρμογής.

Περιληπτικά η φιλοσοφία του συστήματος, είναι η εξής. Ένα Aggregator SPOT εισέρχεται στο δίκτυο και αποστέλλει αίτηση απόκτησης βαρών δικτύου σε μία host εφαρμογή που εκτελείται σε υπολογιστή. Η εφαρμογή μόλις λάβει το αίτημα, δημιουργεί ένα δίκτυο με κάποιες προκαθορισμένες παραμέτρους, το εκπαιδεύει χρησιμοποιώντας ένα Dataset που περιέχει μετρήσεις θερμοκρασίας - υγρασίας και στη συνέχεια αποστέλλει τα συναπτικά βάρη του

εκπαιδευμένου νευρωνικού δικτύου στο Aggregator. Το Aggregator μόλις λάβει τα συναπτικά βάρη, δημιουργεί δίκτυο όμοιο του επιπέδου διαμόρφωσης και εφαρμόζει σε αυτό τα συναπτικά βάρη που μόλις έλαβε από το επίπεδο Διαμόρφωσης. Μόλις ολοκληρωθεί αυτή η διαδικασία, εισέρχονται στο δίκτυο Samplers τα οποία συλλέγουν μετρήσεις και τις αποστέλλουν στο Aggregator το οποίο τροφοδοτεί τις μετρήσεις στο νευρωνικό του δίκτυο. Έξοδος του νευρωνικού δικτύου είναι η πιθανότητα ύπαρξης φωτιάς και έξοδος του Aggregator τελικά είναι η ύπαρξη η όχι φωτιάς (0 και 1 αντίστοιχα)

3.3.1. Το επίπεδο Διαμόρφωσης.

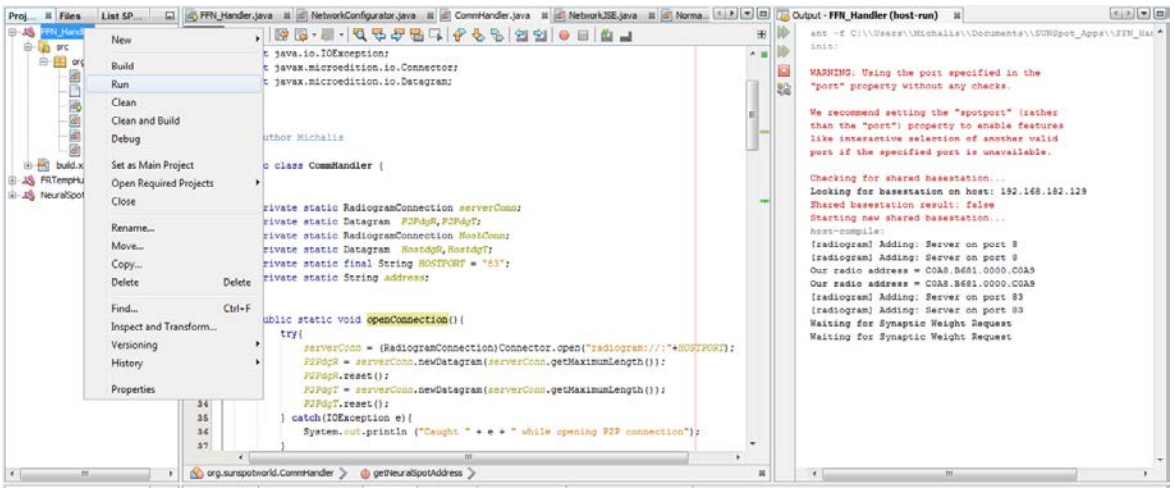
Αυτό το επίπεδο υλοποιείται σε υπολογιστή και αποτελείται από την host εφαρμογή FFN_Handler η οποία, ως host εφαρμογή περιέχει ένα μέρος του API που χρησιμοποιούν τα SunSPOT και ολόκληρη τη βιβλιοθήκη του Encog. Τις βιβλιοθήκες των SunSPOT τις χρησιμοποιεί ώστε να μπορεί να επικοινωνήσει με τα free range SPOT [Aggregator] ενώ τη βιβλιοθήκη του Encog τη χρησιμοποιεί ώστε να κατασκευάσει το νευρωνικό δίκτυο που θα χρησιμοποιηθεί από τα Aggregator SPOT. Όπως είπαμε παραπάνω η host εφαρμογή αποστέλλει εντολές στο basestation SPOT που είναι συνδεδεμένο στον υπολογιστή, προκειμένου να επικοινωνήσει με κάποιο Aggregator SPOT. Κατά την εκκίνηση της, η εφαρμογή ανοίγει μία σύνδεση τύπου Server και περιμένει να λάβει μήνυμα αίτησης αποστολής συναπτικών βαρών δικτύου, από κάποιο Aggregator. Αφού λάβει το αίτημα, αποστέλλει μήνυμα αναγνώρισης [ACK] στο Aggregator, κλείνει τη σύνδεση τύπου Server και ανοίγει καινούργια σύνδεση τύπου Unicast με το συγκεκριμένο Aggregator μόνο. Εν συνέχεια, η εφαρμογή υλοποιεί ένα νευρωνικό δίκτυο και αφού κανονικοποιήσει το Dataset εκπαίδευσης και το Dataset επαλήθευσης, εκπαιδεύει και επαληθεύει το δίκτυο. Εάν κατά την εκπαίδευση πραγματοποιηθούν περισσότερες από 3.000 επαναλήψεις επεξεργασίας του Dataset εκπαίδευσης και το ποσοστό λάθους είναι μεγαλύτερο από την επιτρεπτή τιμή , η διαδικασια εκπαιδευσης διακοπτετε, επαναρχικοποιουνται τα αρχικά-τυχαία βάρη του δικτύου και ξεκινά εκ νέου εκπαίδευση. Μετά το πέρας της εκπαίδευσης του δικτύου ακολουθεί η επαλήθευση του για την οποία χρησιμοποιείται το Dataset επαλήθευσης. Εάν κατά την επαλήθευση δεν παρατηρηθεί ποσοστό λάθους μεγαλύτερο από το επιθυμητό, στα καινούργια απαρουσίαστα μέχρι τώρα στο νευρωνικό

δίκτυο, δεδομένα, τότε αποστέλλονται στο Aggregator SPOT που αιτήθηκε τα συναπτικά βάρη τα εξής :

- Το πλήθος των συναπτικών βαρών. Για παράδειγμα για ένα νευρωνικό δίκτυο $3^{ων}$ επιπέδων με 2,4,1 νευρώνες αντίστοιχα, έχουμε 17 συνδέσεις. 4 από το νευρώνα 1 του πρώτου επιπέδου προς κάθε κόμβο του επόμενου. 4 από το νευρώνα 2 του πρώτου επιπέδου προς κάθε κόμβο του επόμενου επιπέδου. 4 από το νευρώνα πόλωσης του επιπέδου 1 προς κάθε νευρώνα του επόμενου επιπέδου. Στο επόμενο επίπεδο έχουμε από 1 σύνδεση για κάθε ζεύγος - $νευρωνας_i$ προς νευρωνα εξοδου, επομενως 4 και τελος άλλη μία συνδεση από το νευρώνα πόλωσης του επιπέδου 1 προς το νευρώνα εξόδου. Σύνολο 17. Το νούμερο αυτό αποστέλλεται στο Aggregator SPOT για επαλήθευση.
- Τα συναπτικά βάρη του δικτύου της host εφαρμογής αποθηκεύονται σε ένα διάνυσμα και αποστέλλονται μέσω του basestation, στο Aggregator SPOT που έστειλε το αίτημα.

Τέλος η εφαρμογή επανέρχεται σε κατάσταση αναμονής λήψης αιτήματος αποστολής συναπτικών βαρών από κάποιο άλλο Aggregator Spot που πιθανά θα εισέλθει στο δίκτυο αισθητήρων.

Για να εκτελέσουμε την εφαρμογή, στο περιβάλλον ανάπτυξης Netbeans, πατάμε δεξί κλικ στο Project FFN_Handler και run.



Εικόνα 21: Εκτέλεση της host εφαρμογής FFN_Handler από το περιβάλλον ανάπτυξης Netbeans

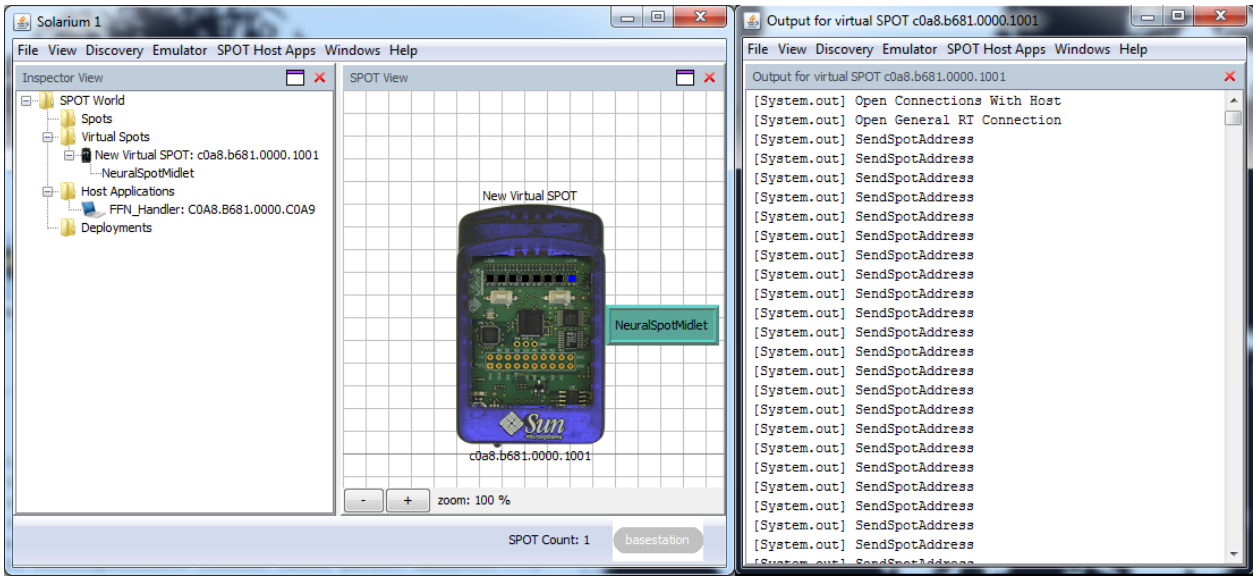
Όπως φαίνεται στην Εικόνα 21 κατά την εκτέλεση της η εφαρμογή ανοίγει μία Servertύπου σύνδεση και αναμένει να λάβει κάποιο αίτημα αποστολής συναπτικών βαρών από κάποιο Aggregator SPOT.

3.3.2. Το επίπεδο Συλλογής και Επεξεργασίας

Σε αυτό το επίπεδο υπάρχει 1 free range SPOT [Aggregator], τοποθετημένο στο χώρο, το οποίο συγκεντρώνει τις μετρήσεις που πραγματοποιούνται από τα SPOT του επιπέδου Δειγματοληψίας. Στην παρούσα εργασία ο μέγιστος αριθμός SPOT για το επίπεδο Δειγματοληψίας είναι 3. Στο επίπεδο Συλλογής και Επεξεργασίας εκτελείται το midlet Aggregator. Για να εκτελέσουμε το midlet ακολουθούμε την παρακάτω διαδικασία:

1. Εκτελούμε την πλατφόρμα ανάπτυξης SPOTManager
2. Από την πλατφόρμα εκτελούμε το Solarium
3. Στο Solarium, στην αριστερή μπάρα επιλογών κάνουμε δεξί κλικ στο διάλογο Virtual Spots και από τον διάλογο που εμφανίζεται επιλέγουμε new virtual spot
4. Στο δεξί πεδίο της εφαρμογής Solarium, στο καινούργιο Spot που εμφανίστηκε, κάνουμε δεξί κλικ και στον καινούργιο διάλογο που εμφανίζετε επιλέγουμε Deploy Midlet
5. Πλοηγούμαστε στον φάκελο που έχουμε αποθηκεύσει το project Aggregator, επιλέγουμε το αρχείο Aggregator.jar και πατάμε OK
6. Κάνουμε ξανά δεξί κλικ στο SPOT που εμφανίζεται στο Solarium και στον διάλογο που εμφανίζετε επιλέγουμε run Midlet -> Aggregator.

Μόλις ενεργοποιηθεί ένα Aggregator SPOT, αναμένει είσοδο από το χρήστη για να καθορίσει το πλήθος εισόδων τις οποίες θα χειριστεί το συγκεκριμένο Aggregator (δηλαδή πόσα Samplers θα στέλνουν μετρήσεις, σε αυτό). Η εισαγωγή του πλήθους των εισόδων πραγματοποιείται με χρήση του πλήκτρου SW1 και οι διαθέσιμες επιλογές πλήθους εισόδων είναι 1, 2 και 3. Αφού γίνει η επιλογή πλήθους εισόδων, η επιλογή καταχωρείται στο Aggregator με χρήση του πλήκτρου SW2. Μόλις γίνει η εισαγωγή του πλήθους εισόδων, το midlet, δεδομένου ότι η διεύθυνση του Server είναι άγνωστη αυτή τη χρονική στιγμή, ανοίγει μία σύνδεση τύπου Broadcast () και αποστέλλει αίτημα απόκτησης συναπτικών βαρών για το νευρωνικό δίκτυο που θα υλοποιήσει. Επαναλαμβάνει την αποστολή αίτησης, όπως φαίνεται στην Εικόνα 22, μέχρι να λάβει μήνυμα αναγνώρισης (ACK) από την host εφαρμογή.



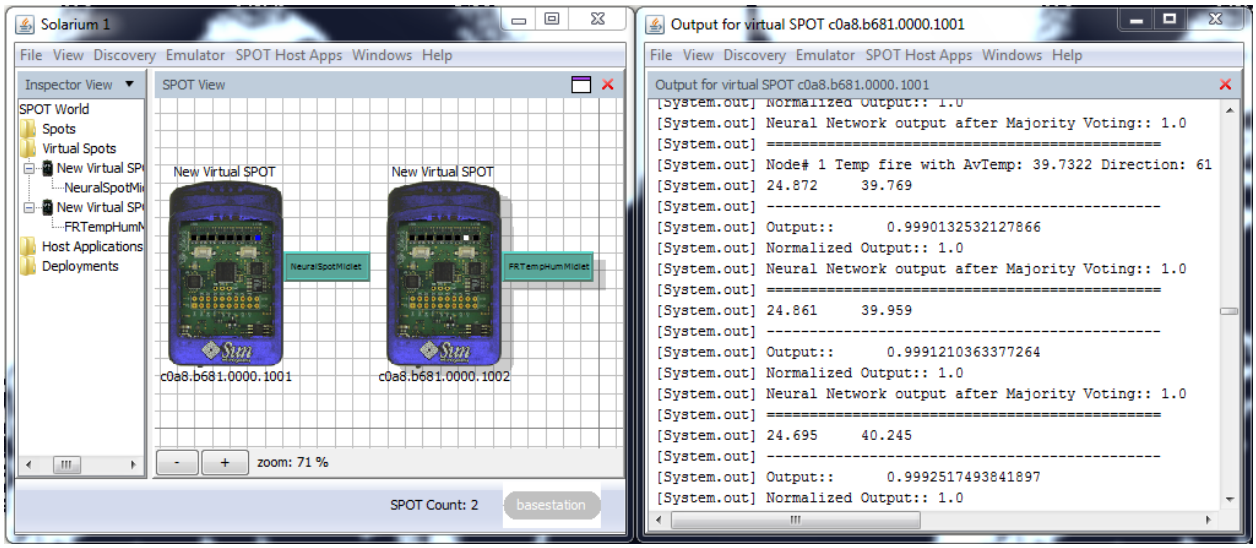
Εικόνα 22. Επανάληψη αίτησης απόκτησης συναπτικών βαρών

Μόλις πραγματοποιηθεί λήψη μηνύματος αναγνώρισης από την host εφαρμογή [Server], το midlet κλείνει την Broadcast σύνδεση και ανοίγει εκ νέου μία "αφοσιωμένη" (dedicated) σύνδεση τύπου Unicast με το basestation της host εφαρμογής. Στη συνέχεια κατασκευάζετε ένα νευρωνικό δίκτυο όμοιο του νευρωνικού δικτύου της host εφαρμογής (ίδιος αριθμός επιπέδων, ίδιος αριθμός νευρώνων σε κάθε επίπεδο, ίδιος πλήθος νευρώνων πόλωσης, ίδιες συναρτήσεις ενεργοποίησης) αλλά δεν το εκπαιδεύει. Αντίθετα περιμένει να λάβει τα βάρη του εκπαιδευμένου νευρωνικού δικτύου από την host εφαρμογή. Πρώτα λαμβάνει από την εφαρμογή το πλήθος των βαρών για επαλήθευση και στη συνέχεια λαμβάνει το διάνυσμα με τα συναπτικά βάρη. Αφού λάβει τα βάρη, τα φορτώνει στο νευρωνικό δίκτυο που έχει υλοποιήσει και το επαληθεύει με το Dataset επαλήθευσης που βρίσκεται στο μονοπάτι /Aggregator/resources[1]. Η διαδικασία της επαλήθευσης είναι ορατή στο χρήστη από το παράθυρο εξόδου του midlet. Μετά τη λήψη των βαρών, το Aggregator δεν θα επικοινωνήσει ξανά με την host εφαρμογή παρά μόνο με τα Samplers του επιπέδου Δειγματοληψίας, επομένως η "αφοσιωμένη" σύνδεση με το basestation της host εφαρμογής, τερματίζεται και ξεκινά καινούργια σύνδεση. Σε αυτή τη σύνδεση το Aggregator αναλαμβάνει το ρόλο του Server και τα Sampler SPOT αναλαμβάνουν το ρόλο των clients. Σε αυτό το σημείο το midlet αναμένει να λάβει μήνυμα από κάποιο

[1] .Ο φάκελος /resources περιέχει τον υποφάκελο /META-INF ο οποίος περιέχει πληροφορίες που χρησιμοποιεί το Squawk VM[3] ώστε να εκτελέσει το midlet. Εκτός από τον υποφάκελο META-INF, μπορεί να περιέχει και οποιαδήποτε άλλη πληροφορία ορισμένη από τον σχεδιαστή, η οποία θα είναι διαθέσιμη στο midlet κατά την εκτέλεση του midlet. Στον φάκελο αυτό έχει αποθηκευθεί και το Dataset επαλήθευσης.

Sampler. Μόλις λάβει μήνυμα, το Aggregator, είτε εισάγει το Sampler SPOT στη λίστα με τις γνωστές σε αυτό συσκευές (εάν είναι η πρώτη φορά που το Sampler SPOT αποστέλλει μήνυμα στο Aggregator), είτε αποθηκεύει τις μετρήσεις που μόλις έλαβε από το Sampler σε μία ουρά 10 θέσεων (κάθε Sampler έχει τη δική του ουρά 10 θέσεων). Ένα Task με συχνότητα εκτέλεσης ρυθμιζόμενη από το χρήστη, ελέγχει εάν οι τιμές των μετρήσεων που υπάρχουν διαθέσιμες στην ουρά, τη στιγμή της εκτέλεσης του, είναι μεγαλύτερες από κάποιες στατιστικές τιμές κατωφλίου (μέγιστη τιμή θερμοκρασίας, φορά μεταβολής θερμοκρασίας, τυπική απόκλιση). Σε περίπτωση που διαπιστωθεί υπέρβαση τιμής κατωφλίου από κάποια μέτρηση, τότε το τελευταίο ζεύγος θερμοκρασίας-υγρασιας από την ουρά 10 θέσεων για κάθε Sampler που είναι συνδεδεμένο με το Aggregator και οι επόμενες 9 καινούργιες μετρήσεις, τροφοδοτούνται στο νευρωνικό δίκτυο, όπως φαίνεται στην Εικόνα 23. Εάν , κατά τη λειτουργία του νευρωνικού δικτύου, διαπιστωθεί ότι πρόκειται για εσφαλμένο συναγερμό, τότε παύει η τροφοδοσία μετρήσεων στο νευρωνικό δίκτυο. Για παράδειγμα, μπορεί μία μέτρηση σε κάποιο Sampler να δείξει peek στη θερμοκρασία περιβάλλοντος (για οποιοδήποτε λόγο) με ένδειξη π.χ. 27°C αντί της πραγματικής των 22°C. Αυτή η απότομη αύξηση στη θερμοκρασία θα οδηγήσει σε απότομη αύξηση της τυπικής απόκλισης των μετρήσεων και θα πυροδοτήσει την τροφοδότηση του νευρωνικού δικτύου. Εάν οι επόμενες μετρήσεις επιστρέψουν στο φυσιολογικό επίπεδο (το επίπεδο πριν την απότομη αύξηση θερμοκρασίας), η τυπική απόκλιση θα ελαττωθεί, επομένως σταματάει η τροφοδοσία μετρήσεων στο νευρωνικό δίκτυο.

Εάν οι μετρήσεις δεν επιστρέψουν στο κανονικό επίπεδο και συνεχίσουν να κυμαίνονται στους 27°C, η τυπική απόκλιση θα ελαττωθεί, αλλά θα αυξηθεί ο μέσος όρος θερμοκρασίας ξεπερνώντας την τιμή κατωφλίου για τη θερμοκρασία επομένως πάλι θα πυροδοτηθεί η τροφοδοσία του νευρωνικού δικτύου.



Εικόνα 23. Τροφοδοσία του νευρωνικού δικτύου με μετρήσεις που υπερβαίνουν τις τιμές κατωφλίου.

Όπως φαίνεται στην Εικόνα 23, για κάθε είσοδο στο νευρωνικό δίκτυο, αναγράφεται στο παράθυρο εξόδου του Aggregator:

- ο λόγος για τον οποίο έγινε τροφοδοσία μετρήσεων στο δίκτυο [εδώ σημειώνεται μέση θερμοκρασία 39.7322°C με τιμή κατωφλίου τους 25°C]
- η φορά την οποία ακολουθεί το μέγεθος της θερμοκρασίας [εδώ, η τιμή 61 υποδηλώνει ότι η τιμή της θερμοκρασίας έχει ανοδική πορεία στις τελευταίες 61 μετρήσεις],
- το ζεύγος μετρήσεων που τροφοδοτήθηκε στο δίκτυο,
- η έξοδος του νευρωνικού δικτύου και
- η τελική έξοδος του συστήματος μετά από κανονικοποίηση της εξόδου του νευρωνικού δικτύου και η “ψήφιση πλειοψηφίας” [Majority Voting [13]].

Στο συγκεκριμένο παράδειγμα το αποτέλεσμα του majority voting είναι πάντα ίσο με 1, επειδή έχουμε 1 είσοδο και 1 έξοδο. Πρέπει να σημειωθεί εδώ ότι το πλήθος εισόδων που επιλέγεται για το νευρωνικό δίκτυο δεν έχει επίδραση στην αρχιτεκτονική του δικτύου. Το νευρωνικό δίκτυο που υλοποιείται τόσο στην host εφαρμογή όσο και στο midlet του επιπέδου Συλλογής και Επεξεργασίας είναι πάντα της μορφής:

- Επίπεδο εισόδου με 2 νευρώνες (ένας για τη θερμοκρασία και ένας για την υγρασία)
- Κρυφό επίπεδο με 4 νευρώνες
- Επίπεδο εξόδου με 1 νευρώνα και
- Νευρώνες πόλωσης στα επίπεδα εισόδου και κρυφό επίπεδο.

Αυτό που αλλάζει ανάλογα με το πλήθος των εισόδων είναι ο τρόπος με τον οποίο χειριζόμαστε το νευρωνικό δίκτυο. Στην περίπτωση της μίας εισόδου (1 ζεύγος μετρήσεων θερμοκρασίας/υγρασίας) τροφοδοτούμε το νευρωνικό δίκτυο με το ζεύγος μετρήσεων και παίρνουμε την έξοδο. Σε περίπτωση περισσότερων της μίας εισόδου, κατασκευάζεται διάνυσμα με τα n ζεύγη μετρήσεων (όπου n το πλήθος των Samplers που αποστέλλουν μετρήσεις) το οποίο τροφοδοτείται σειριακά στο νευρωνικό δίκτυο. Για κάθε είσοδο από το διάνυσμα εισόδου, η έξοδος του νευρωνικού δικτύου αποθηκεύεται σε ένα διάνυσμα και η τελική έξοδος του νευρωνικού δικτύου θεωρείται τελικά, ολόκληρο το διάνυσμα.

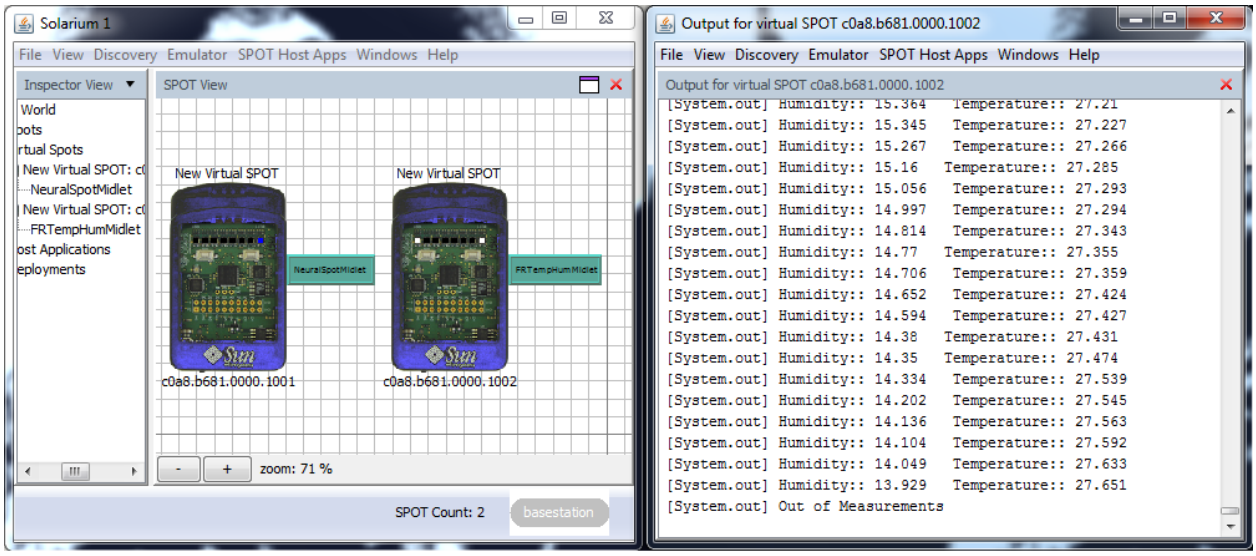
3.3.3. Το επίπεδο Δειγματοληψίας.

Σε αυτό το επίπεδο υπάρχουν από 1 μέχρι n (εδώ $n_{max} = 3$) free range SPOT, τοποθετημένα στο χώρο, τα οποία πραγματοποιούν μετρήσεις θερμοκρασίας και υγρασίας και τις αποστέλλουν στο επόμενο επίπεδο. Τα SPOT αυτού του επιπέδου εκτελούν το midlet Sampler. Τις μετρήσεις που αποστέλλουν, δεν τις λαμβάνουν από το περιβάλλον (καθώς πρόκειται για εικονικά SPOT σε περιβάλλον εξομοίωσης) αλλά από ένα αρχείο που περιέχει μετρήσεις θερμοκρασίας/υγρασίας Υπάρχουν 3 διαφορετικά αρχεία με διαφορετικά σενάρια/μετρήσεις το κάθε ένα προκειμένου να γίνει μία ικανοποιητική προσομοίωση ενός πραγματικού δασικού περιβάλλοντος. Κάθε SPOT αυτού του επιπέδου κατά την εκκίνηση του αναμένει είσοδο από το χρήστη προκειμένου να καθοριστεί, από ποιο αρχείο θα λαμβάνει τις μετρήσεις που θα αποστέλλει στο επίπεδο Συλλογής και Επεξεργασίας. Η επιλογή του αρχείου γίνεται από το πλήκτρο SW1 και η αποθήκευση της επιλογής από το κουμπί SW2. Τα βήματα που ακολουθούνται στη συνέχεια είναι τα εξής:

1. Ξεκινά μία “αφοσιωμένη” σύνδεση, η οποία έχει συγκεκριμένη διεύθυνση, αυτή του Aggregator SPOT στου οποίου την εποπτεία ανήκει το Sampler SPOT.
2. Αποστέλλεται στο Aggregator ένα κενό "αναγνωριστικό" μήνυμα από το Sampler SPOT.. Μόλις ο Aggregator λάβει αυτό το μήνυμα εισάγει το Sampler SPOT στη λίστα με τις γνωστές σε αυτό συσκευές και αποστέλλει στο Sampler μήνυμα αναγνώρισης [ACK].
3. Μόλις το Sampler λάβει το μήνυμα αναγνώρισης [ACK], αποθηκεύει σε ένα προσωρινό String το αρχείο σεναρίου / μετρήσεων που επιλέχτηκε από το χρήστη και βρίσκεται στο φάκελο /resources.

4. Ξεκινά ένα task με συχνότητα εκτέλεσης 500ms, το οποίο διαβάζει ένα ζεύγος τιμών (θερμοκρασία - υγρασία) από το προσωρινό String και στη συνέχεια το αποστέλλει στο Aggregator μέσω της σύνδεσης του βήματος 1.
5. Όταν αποστείλει όλες τις μετρήσεις που περιέχονται στο αρχείο που έχει επιλεχθεί, τερματίζει το task και ενημερώνει ότι δεν έχει διαθέσιμες άλλες μετρήσεις.

Στην Εικόνα 24. εμφανίζεται το παράθυρο εξόδου ενός Sampler κατά τη διάρκεια εκτέλεσης του midlet καθώς και ο τερματισμός της εν λόγω διαδικασίας με το μήνυμα “Out of measurements” στην τελευταία γραμμή (όταν πλέον τελειώσουν οι μετρήσεις που περιέχονται στο αρχείο).



Εικόνα 24: Η έξοδος ενός Sampler SPOT στο παράθυρο εξόδου του Solarium

3.4. Μελέτη - Δοκιμές και Ενέργειες

Όπως ‘ήδη αναφέρθηκε στην παράγραφο 2.2, η επιτυχής υλοποίηση ενός νευρωνικού δικτύου επιτυγχάνεται μέσα από δοκιμές και αλλαγές στην αρχιτεκτονική του και τις παραμέτρους του. Εκτός από τις δοκιμές που πραγματοποιήθηκαν, χρειάστηκε να γίνουν port κάποιες κλάσεις του αρχικού Encog σε j2me. Στα επόμενα κεφάλαια αναλύονται ο ενέργειες και δόκιμες που πραγματοποιήθηκαν προκειμένου να υλοποιηθεί ένα αποδοτικό νευρωνικό δίκτυο σε SPOT.

3.4.1.Τροποποίηση Κλάσεων [Porting]

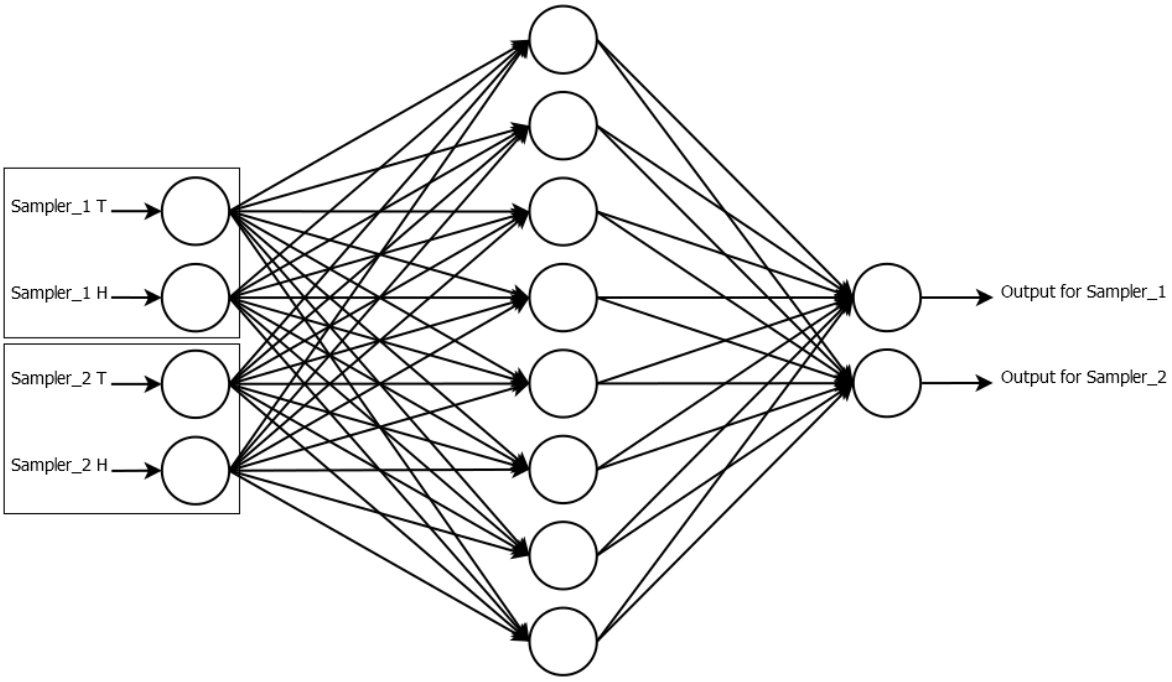
Το αρχικό port του Encog(JSE) στο EncogME(j2me), έγινε με σκοπό να υλοποιήσει ένα απλό feed forward δίκτυο για πειραματικούς σκοπούς. Η ορθή λειτουργία του δικτύου επαληθεύτηκε με τη μοντελοποίηση ενός τελεστή XOR με δυο εισόδους και εύρος τιμών [0 - 1] για την είσοδο, έτσι δεν χρειάστηκε να υλοποιηθούν κλάσεις που κανονικοποιούν την είσοδο. Στο δίκτυο της παρούσας εργασίας οι μετρήσεις που διατίθενται ως είσοδος δεν έχουν εύρος τιμών [0 - 1] αλλά έχουν τιμές από 0 έως 100 για την υγρασία και αρνητικές ή θετικές για τη θερμοκρασία[2]. Όπως είπαμε και σε προηγούμενο κεφάλαιο η παροχή μη κανονικοποιημένων τιμών στο νευρωνικό δίκτυο δεν είναι απαγορευτική, όμως ένα νευρωνικό δίκτυο είναι πιο αποδοτικό όταν η είσοδος του είναι κανονικοποιημένη. Γι αυτό το λόγο τροποποιήθηκαν οι κλάσεις NormalizationAction και NormalizedField. Επίσης για δοκιμαστικούς και μόνο λόγους τροποποιήθηκε η κλάση ActivationTANH, η οποία υλοποιεί τη συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης. Τέλος προστέθηκε στη βιβλιοθήκη EncogME η κλάση REAL [14] η οποία στην ουσία είναι μία βιβλιοθήκη μαθηματικών για JavaME. Ο λόγος για τον οποίο προστέθηκε η βιβλιοθήκη είναι ότι η κλάση Math του EncogME, δεν υλοποιεί σωστά συναρτήσεις όπως η εκθετική με αποτέλεσμα το νευρωνικό δίκτυο να μην συμπεριφέρεται όπως αναμένεται.

[2] Το ακριβές εύρος τιμών της θερμοκρασίας είναι άγνωστο όμως ξέρουμε σίγουρα ότι τα ηλεκτρονικά του συστήματος (ή οποιουδήποτε συστήματος με εξαίρεση συστήματα με ηλεκτρονικά στρατιωτικών προδιαγραφών) από τους 85°C και πάνω και -15°C και κάτω σταματούν να λειτουργούν σωστά. Οι μετρήσεις στο Dataset που έχουμε, έχουν εύρος [5 - 60]. Επομένως κρίθηκε κατάλληλο να κανονικοποιήσουμε στο υπαρκτό εύρος και όχι στο θεωρητικό [(-15) - 85].

3.4.2. Αρχιτεκτονική Δικτύου. Δίκτυο μονής/πολλαπλής εισόδου

Η είσοδος στο νευρωνικό δίκτυο που υλοποιήθηκε αποτελείται από ένα ζεύγος μετρήσεων υγρασίας και θερμοκρασίας που λαμβάνονται από ένα Sampler SPOT. Σε περίπτωση που υπάρχουν παραπάνω από ένα Sampler SPOT διακρίνονται δυο επιλογές.

Η πρώτη είναι να κατασκευαστεί δίκτυο πολλαπλής εισόδου, διάστασης ίσης με το πλήθος των Sampler που αναφέρονται στο συγκεκριμένο δίκτυο. Συγκεκριμένα εάν στο Aggregator SPOT, στέλνουν δεδομένα 2 Sampler SPOT, τότε το Aggregator κατασκευάζει νευρωνικό δίκτυο με διάσταση εισόδου ίση με το πλήθος των Sampler. Στην περίπτωση αυτή το δίκτυο δέχεται συνολικά 2 μετρήσεις υγρασίας και 2 μετρήσεις θερμοκρασίας. Επομένως στο επίπεδο εισόδου του έχει από 1 νευρώνα για κάθε μέτρηση θερμοκρασίας από τις 2 που δέχεται και από 1 νευρώνα για κάθε μέτρηση υγρασίας από τις 2 που επίσης δέχεται. Αντίστοιχα στο επίπεδο εξόδου δεν έχει ένα μόνο νευρώνα αλλά 2. Ουσιαστικά δηλαδή έχει 1 έξοδο για κάθε είσοδο που δέχεται.



Εικόνα 25. Νευρωνικό δίκτυο αρχιτεκτονικής πολλαπλής εισόδου

Η αρχιτεκτονική αυτή έχει αρκετά μειονεκτήματα με κυριότερο ότι η πολυπλοκότητα του δικτύου αυξάνει απαγορευτικά δεδομένου ότι το δίκτυο υλοποιείται σε επεξεργαστή

περιορισμένης υπολογιστικής ισχύος ο οποίος τροφοδοτείται από μπαταρία. Συγκεκριμένα και δεδομένου ότι έχουμε επιλέξει να χρησιμοποιήσουμε κρυφό επίπεδο με πλήθος νευρώνων ίσο με το διπλάσιο του πλήθους των νευρώνων του επιπέδου εισόδου (ο λόγος για τον οποίο έγινε αυτή η επιλογή θα εξηγηθεί σε επόμενο κεφάλαιο), το πλήθος των συναπτικών βαρών που χρησιμοποιούνται στο δίκτυο προκύπτει από την εξίσωση:

$$S_{sw} = \left(nC_{inp} * nC_{hL}\right) + \left(nC_{outp} * nC_{hL}\right) + \left(nC_{hL} * bnC_{inp}\right) + \left(nC_{outp} * bnC_{hL}\right) \quad [4.2.1]$$

Όπου:

- S_{sw} είναι το πλήθος των συναπτικών βαρών του δικτύου
- nC_{inp} ειναι το πληθος των νευρωνων του επιπεδου εισοδου
- nC_{hL} ειναι το πληθος των νευρονων του κρυφου επιπεδου
- nC_{outp} ειναι το πληθος των νευρωνων του επιπεδου εξοδου και
- bnC_{inp} και bnC_{hL} είναι το πλήθος των νευρώνων πόλωσης του επιπέδου εισόδου και του κρυφού επιπέδου αντιστοίχως

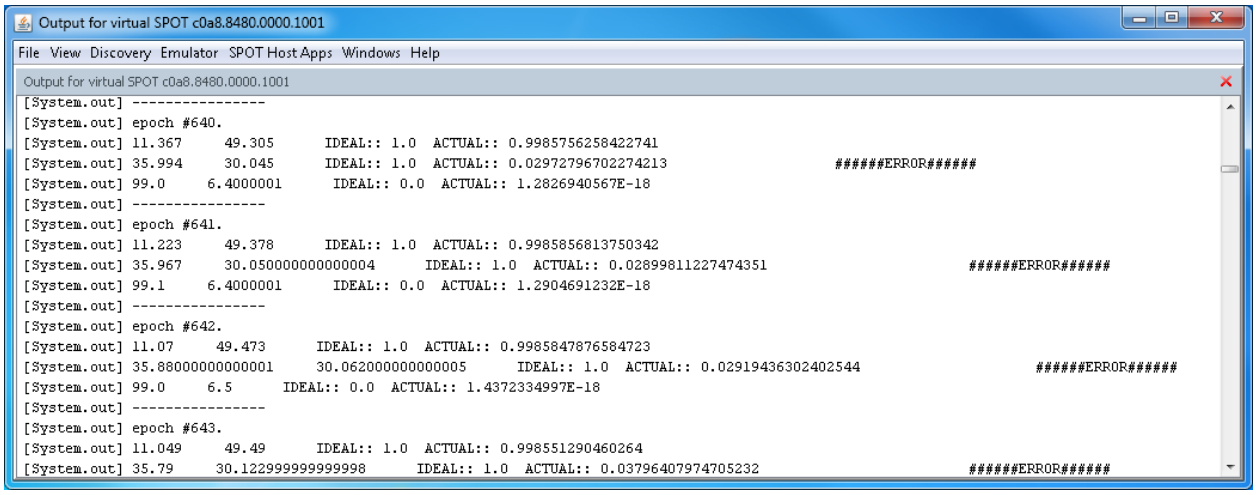
Το πλήθος των πράξεων που εκτελούνται στο δίκτυο από τη στιγμή που αυτό δέχεται ένα σήμα εισόδου μέχρι να υπάρξει σήμα εξόδου προκύπτει από την εξίσωση:

$$\left([(2n+2)*2n] + \left[(4n+2)*\frac{n}{2}\right]\right) = 6n^2 + 5n \quad [4.2.2]$$

όπου n το πλήθος των νευρώνων εισόδου.

Από την πιο πάνω εξίσωση [4.2.1] προκύπτει πως για ένα νευρωνικό δίκτυο μίας εισόδου (2 νευρώνες επιπέδου εισόδου) έχουμε $(2*4)+(1*4)+(4*1)+(1*1) = 17$ συναπτικα βαρη ενώ για ένα δίκτυο 4 εισόδων (8 νευρώνες στο επίπεδο εισόδου) έχουμε $(8*16)+(4*16)+(16*1)+(4*1) = 212$ συναπτικά βάρη.

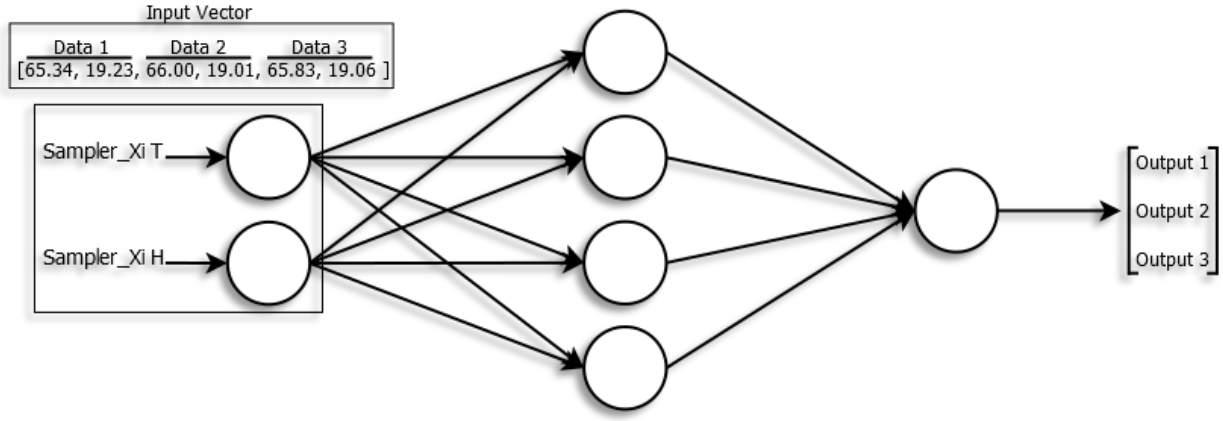
Άλλο ένα μειονέκτημα αυτής της αρχιτεκτονικής είναι ότι υπάρχει μεγάλη αλληλεξάρτηση μεταξύ των καναλιών εξόδου. Όπως φαίνεται και στην Εικόνα 26, ενώ το πρώτο κανάλι εξόδου, παρέχει την αναμενόμενη έξοδο το δεύτερο κανάλι εξόδου δεν παρέχει σωστή έξοδο δίνοντας για 35.967 βαθμούς υγρασίας και 30.05 βαθμούς θερμοκρασίας πιθανότητα ίση με 0.028% να υπάρχει πυρκαγιά.



Εικόνα 26. Έξοδος νευρωνικού δικτύου με είσοδο/εξοδο τριών στοιχείων

Σταδιακά, η τιμή της εξόδου στο δεύτερο κανάλι θα φτάσει στο επιθυμητό επίπεδο, όμως ο αριθμός μετρήσεων που θα δώσουν εσφαλμένο αποτέλεσμα αποτελεί σημαντικό ποσοστό του συνόλου των μετρήσεων που σηματοδοτούν πυρκαγιά. Αυτή η καθυστέρηση στην απόκριση του δεύτερου καναλιού οφείλεται στο γεγονός ότι το τρίτο κανάλι έχει ως έξοδο 0. Αντίστοιχα το τρίτο κανάλι εξόδου επηρεάζεται από το δεύτερο κανάλι που σημαίνει ότι εάν το δεύτερο κανάλι εξόδου δίνει έξοδο ίση με 0 τότε μία αλλαγή στο κανάλι 3 από το 0 στο 1 θα εμφανιστεί στην έξοδο με καθυστέρηση. Επίσης παρατηρήθηκε ότι πολλές φορές κάποιο κανάλι που δίνει έξοδο ίση με 1 επηρεάζει κάποιο γειτονικό κανάλι που δίνει έξοδο ίση με 0 προκαλώντας ανοδική πορεία της εξόδου αυτής προς το 1. Γενικά μπορούμε να πούμε ότι υπάρχει μεγάλη αλληλεξάρτηση μεταξύ γειτονικών καναλιών, γεγονός το οποίο κάνει το δίκτυο αρκετά "δυσκίνητο" και με μεγάλο ποσοστό σφαλμάτων εξόδου. Ένα επιπλέον μειονέκτημα της συγκεκριμένης αρχιτεκτονικής είναι ότι η εκπαίδευση του δικτύου γίνεται περισσότερο χρονοβόρα όπως επίσης ότι χρειάζεται κατασκευή Dataset εκπαίδευσης - επαλήθευσης, για κάθε περίπτωση μεγέθους εισόδου.

Η επόμενη επιλογή ως προς την αρχιτεκτονική του δικτύου αποτυπώνεται στην Εικόνα 27 και συνίσταται στην κατασκευή ενός δικτύου με μονή είσοδο - μονή έξοδο, ανεξάρτητα από το μέγεθος εισόδου, στο οποίο σε περίπτωση εισόδου μεγαλύτερης από 1, τα ζεύγη τιμών παρουσιάζονται στο δίκτυο σειριακά από ένα προσωρινό διάνυσμα.



Εικόνα 27: Νευρωνικό δίκτυο μονής εισόδου μονής εξόδου. Η είσοδος τροφοδοτείται σειριακά με ζεύγη μετρήσεων και η έξοδος για κάθε ζεύγος μετρήσεων εισόδου αποθηκεύεται σε ένα διάνυσμα. Έξοδος του δικτύου θεωρείται ολόκληρο το διάνυσμα

Παραδείγματος χάριν σε περίπτωση που το δίκτυο έχει μέγεθος εισόδου ίσο με 3 τότε κατασκευάζεται ένα δίκτυο με μονή είσοδο μονή έξοδο (2 νευρώνες εισόδου, 1 νευρώνας εξόδου, κρυφό επίπεδο, πλήθος νευρώνων κρυφού επιπέδου ίσο με το διπλάσιο του πλήθους των νευρώνων του επιπέδου εισόδου). Τα τρία ζεύγη τιμών που οδηγούνται στο Aggregator SPOT, αποθηκεύονται σε ένα διάνυσμα έξι θέσεων και στη συνέχεια παρουσιάζονται στο νευρωνικό δίκτυο σειριακά, ένα ζεύγος τη φορά. Αντίστοιχα η έξοδος του νευρωνικού δικτύου για κάθε είσοδο από το διάνυσμα εισόδου, αποθηκεύεται σε ένα διάνυσμα τριών θέσεων το οποίο θεωρείται ως και η τελική έξοδος του δικτύου. Τα πλεονεκτήματα αυτής της αρχιτεκτονικής είναι:

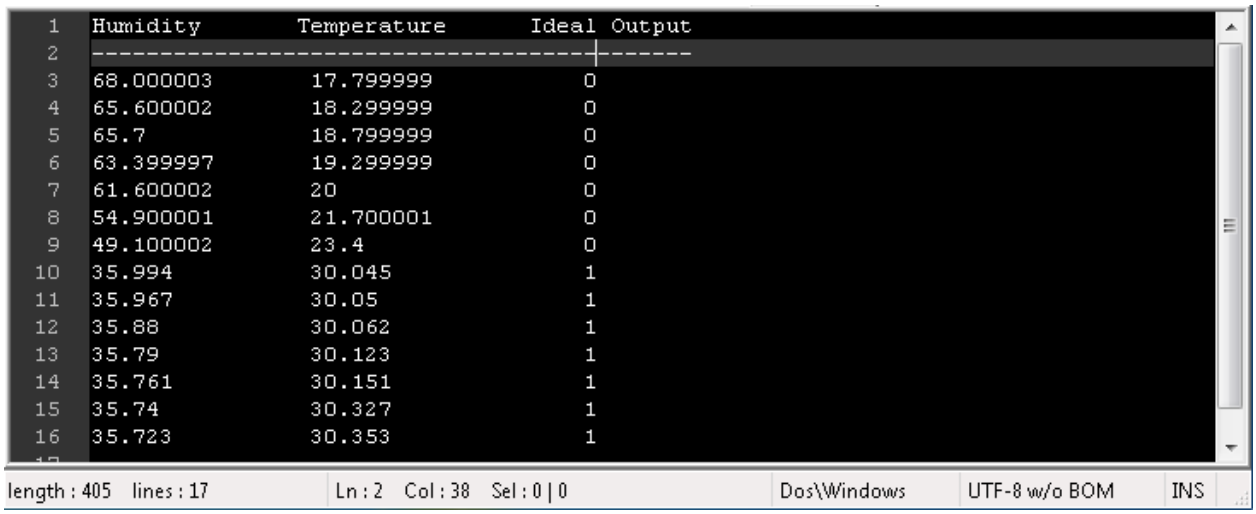
- έχουμε ένα Dataset εκπαίδευσης/επαληθευσης ανεξάρτητα από το μέγεθος εισόδου
- η εκπαίδευση του δικτύου είναι γρηγορότερη και δεν αποτυγχάνει - σε αντίθεση με την εκπαίδευση δικτύου με πολλαπλή είσοδο η οποία πολλές φορές απαιτεί πολλά σετ επαναλήψεων του Dataset προκειμένου να προσδιοριστούν τα βάρη του δικτύου, με υπολογίσιμη πιθανότητα να υπάρξει εμπλοκή με κάποιο τοπικό ελάχιστο (local minima)
- υπάρχει ένα κανάλι εξόδου το οποίο δεν επηρεάζεται από τις επόμενες εξόδους
- το δίκτυο που προκύπτει είναι μικρό και δεν απαιτεί υπολογίσιμη υπολογιστική ισχύ ή αποθηκευτικό χώρο, ανεξάρτητα από το μέγεθος εισόδου
- τέλος, έχει μεγαλύτερη επεκτασιμότητα. Για παράδειγμα ένα δίκτυο αυτής της αρχιτεκτονικής με μέγεθος εισόδου ίσο με 25 (25 Samplers αποστέλλουν μετρήσεις στο Aggregator SPOT), έχει 2 νευρώνες επιπέδου εισόδου και για να προκύψει μία έξοδος

εκτελεί 34 πράξεις (με βάσει την εξίσωση [4.2.2] ανωτέρω) ή για 25 εξόδους $25 * 34 = 850$ πραξεις. Αντίστοιχα ένα νευρωνικό δίκτυο πολλαπλής εισόδου με μέγεθος εισόδου 25, έχει συνολικά 50 νευρώνες εισόδου και με βάση την εξίσωση [4.2.2] προκειμένου να διαθέσει μία έξοδο εκτελεί $\left((6 * 50^2) + (5 * 50)\right) = 15.250$ πράξεις.

Είναι φανερό πως η δεύτερη αρχιτεκτονική είναι καταλληλότερη της πρώτης δεδομένων τόσο των σχεδιαστικών επιλογών που έγιναν ως προς τη δομή του δικτύου όσο και της πλατφόρμας (περιορισμένης επεξεργαστική ισχύος και τροφοδοσίας) στην οποία καλούμαστε να υλοποιήσουμε το νευρωνικό δίκτυο.

3.4.3. Dataset. Μορφοποίηση. Επίδραση στο Δίκτυο

Το Dataset, όπως είδαμε και στην παράγραφο 2.4.1, είναι ένα σύνολο κατάλληλων μετρήσεων (θερμοκρασίας, υγρασίας κ.λπ.) το οποίο χρησιμοποιείται για την εκπαίδευση του νευρωνικού δικτύου. Το Dataset που χρησιμοποιήθηκε περιέχει μετρήσεις υγρασίας και θερμοκρασίας που ελήφθησαν σε διάρκεια μίας εβδομάδας με μέσο όρο συχνότητας δειγματοληψίας 2 μετρήσεων ανά ώρα. Στο εν λόγω Dataset δεν περιέχονται μετρήσεις από πραγματική πυρκαγιά. Οι τιμές για την μέτρηση της υγρασίας κυμαίνονται στο εύρος [28.6 - 99.8] ενώ αυτές της μέτρησης θερμοκρασίας στο εύρος [5.199°C - 29.799°C]. Δεδομένου ότι στο νευρωνικό δίκτυο κατά τη διάρκεια της εκπαίδευσης του πρέπει να τροφοδοτηθούν δεδομένα που να καλύπτουν όλες τις πιθανές περιπτώσεις λειτουργίας, αποφασίστηκε τα ζεύγη μετρήσεων που έχουν: τιμή για τη θερμοκρασία πάνω από 23°C και για την υγρασία κάτω από 35% να σηματοδοτούν πυρκαγιά. Η επιλογή αυτή αποτυπώνεται στην Εικόνα 28. Έτσι το εύρος μη ύπαρξης πυρκαγιάς είναι [35% - 100%] για την υγρασία και [5°C - 22.99°C] για τη θερμοκρασία ενώ το εύρος ύπαρξης πυρκαγιάς είναι [28.6% - 35%] για την υγρασία και [23°C - 29.799°C] για τη θερμοκρασία.

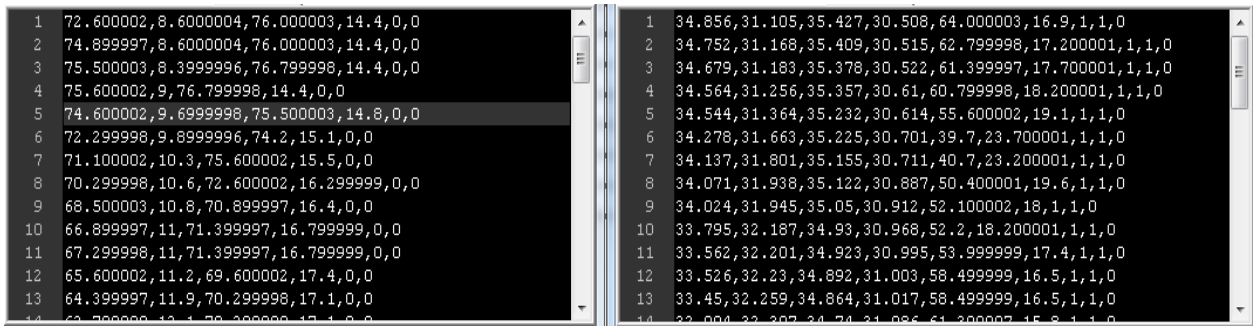


Humidity	Temperature	Ideal	Output
68.000003	17.799999	0	
65.600002	18.299999	0	
65.7	18.799999	0	
63.399997	19.299999	0	
61.600002	20	0	
54.900001	21.700001	0	
49.100002	23.4	0	
35.994	30.045	1	
35.967	30.05	1	
35.88	30.062	1	
35.79	30.123	1	
35.761	30.151	1	
35.74	30.327	1	
35.723	30.353	1	

length : 405 lines : 17 | Ln : 2 Col : 38 Sel : 0 | 0 | Dos\Windows | UTF-8 w/o BOM | INS

Εικόνα 28: Ο διαχωρισμός ύπαρξης και μη ύπαρξης πυρκαγιάς

Παράλληλα κατασκευάστηκε Dataset που αντιστοιχεί στη δομή του δικτύου με μέγεθος εισόδου 2 και Dataset που αντιστοιχεί στη δομή δικτύου με μέγεθος εισόδου ίση με 3 (Εικόνα 29).



```
72.600002,8.6000004,76.000003,14.4,0,0
74.899997,8.6000004,76.000003,14.4,0,0
75.500003,8.3999996,76.799998,14.4,0,0
75.600002,9,76.799998,14.4,0,0
74.600002,9.6999998,75.500003,14.8,0,0
72.299998,9.8999996,74.2,15.1,0,0
71.100002,10.3,75.600002,15.5,0,0
70.299998,10.6,72.600002,16.299999,0,0
68.500003,10.8,70.899997,16.4,0,0
66.899997,11,71.399997,16.799999,0,0
67.299998,11,71.399997,16.799999,0,0
65.600002,11.2,69.600002,17.4,0,0
64.399997,11.9,70.299998,17.1,0,0
```

```
34.856,31.105,35.427,30.508,64.000003,16.9,1,1,0
34.752,31.168,35.409,30.515,62.799998,17.200001,1,1,0
34.679,31.183,35.378,30.522,61.399997,17.700001,1,1,0
34.564,31.256,35.357,30.61,60.799998,18.200001,1,1,0
34.544,31.364,35.232,30.614,55.600002,19.1,1,1,0
34.278,31.663,35.225,30.701,39.7,23.700001,1,1,0
34.137,31.801,35.155,30.711,40.7,23.200001,1,1,0
34.071,31.938,35.122,30.887,50.400001,19.6,1,1,0
34.024,31.945,35.05,30.912,52.100002,18,1,1,0
33.795,32.187,34.93,30.968,52.2,18.200001,1,1,0
33.562,32.201,34.923,30.995,53.999999,17.4,1,1,0
33.526,32.23,34.892,31.003,58.499999,16.5,1,1,0
33.45,32.259,34.864,31.017,58.499999,16.5,1,1,0
```

Εικόνα 29: Datasets σε μορφή CSV. Αριστερά Dataset για μέγεθος εισόδου ίσο με 2 και δεξιά για μέγεθος εισόδου ίσο με 3

Αρχικά τα Dataset περιείχαν λίγες μετρήσεις στην περιοχή της πυρκαγιάς με αποτέλεσμα το δίκτυο να μην συγκλίνει κατά τη διάρκεια της εκπαίδευσης. Γι αυτόν το λόγο και προκειμένου να επιτευχθεί η εκπαίδευση του δικτύου, προστέθηκαν πλασματικές τιμές στην περιοχή πυρκαγιάς (δηλαδή μετρήσεις στο εύρος [28.6% - 35%] για την υγρασία και [23°C - 29.799°C] για τη θερμοκρασία). Παρατηρήθηκε ότι ένα λειτουργικά αξιοποιήσιμο Dataset είναι το Dataset το οποίο περιέχει μετρήσεις από όλες τις περιοχές μετρήσεων με παρόμοια ποσόστωση για κάθε περιοχή. Η απόδοση του νευρωνικού δικτύου κρίνεται ικανοποιητική με αυτά τα εύρη τιμών παρουσιάζεται όμως το πρόβλημα του ότι τα συγκεκριμένα εύρη δεν είναι ιδιαίτερα ρεαλιστικά.

Στη συνέχεια κατασκευάστηκαν καινούργια Datasets τα οποία περιέχουν μετρήσεις με εύρος [10% - 35%] για την υγρασία και [30°C - 50°C] για τη θερμοκρασία στην περιοχή πυρκαγιάς και [35° - 99.8°] για την υγρασία και [5.199°C - 29.99°C] για τη θερμοκρασία στην περιοχή απουσίας πυρκαγιάς. Επίσης η περιοχή μετρήσεων [10° - 35°] για την υγρασία και [5.199°C - 29.999°C] για τη θερμοκρασία ορίστηκε ως περιοχή απουσίας πυρκαγιάς (ημέρες που ο ατμοσφαιρικός αέρας είναι ξηρός), ενώ η περιοχή μετρήσεων [35° - 99.8°] για την υγρασία και [30°C - 50°C] για τη θερμοκρασία ορίστηκε ως περιοχή ύπαρξης πυρκαγιάς θεωρώντας ότι:

1. Θερμοκρασίες πάνω από 30°C είναι απίθανο να σημειωθούν χωρίς ύπαρξη φωτιάς
2. Είναι πιθανό να υπάρχει πυρκαγιά και ο αισθητήρας θερμοκρασίας είτε να δυσλειτούργησε δίνοντας λανθασμένη τιμή, είτε η υγρασία - καθώς είναι αργά μεταβαλλόμενο μέγεθος - δεν έχει ακόμα πέσει κάτω από την τιμή κατωφλίου

Με τη χρήση αυτού του Dataset το οποίο, αν και δεν περιέχει πραγματικές μετρήσεις που ελήφθησαν κατά τη διάρκεια πυρκαγιάς, είναι περισσότερο ρεαλιστικό από το πρώτο Dataset, παρουσιάστηκε το πρόβλημα αλληλεξάρτησης μεταξύ των καναλιών εξόδου του δικτύου με μέγεθος εισόδου μεγαλύτερο από 1, καθώς το ποσοστό λάθους και ο βαθμός αλληλεπίδρασης μεταξύ γειτονικών καναλιών αυξήθηκε δραματικά. Αυτός ήταν ένας από τους βασικούς λόγους που τελικά επιλέχτηκε η αρχιτεκτονική μονής εισόδου μονής εξόδου έναντι της αρχιτεκτονικής πολλαπλής εισόδου πολλαπλής εξόδου. Πρέπει να σημειωθεί εδώ ότι το Dataset εκπαίδευσης πρέπει να περιέχει όλες τις περιοχές μετρήσεων. Το Dataset που τελικά χρησιμοποιήθηκε στην παρούσα εργασία έχει συνολικά 7583 μετρήσεις. Από αυτές:

- 3233 μετρήσεις δεν σηματοδοτούν πυρκαγιά. Από αυτές 517 είναι στο εύρος [10% - 35%] για την υγρασία και [5.199°C - 29.999°C] για τη θερμοκρασία και οι υπόλοιπες 2716 είναι στο εύρος [35° - 99.8°] για την υγρασία και [5.199°C - 29.99°C] για τη θερμοκρασία.
- 4350 μετρήσεις σηματοδοτούν πυρκαγιά. Από αυτές 1000 είναι στο εύρος [35° - 99.8°] για την υγρασία και [30°C - 50°C] για τη θερμοκρασία και 3350 μετρήσεις είναι στο εύρος [10° - 35°] για την υγρασία και [30°C - 50°C] για τη θερμοκρασία.

3.4.4. Κρυφό Επίπεδο - Αριθμός Νευρώνων Κρυφού Επιπέδου

Όπως είπαμε και στην παράγραφο 2.2, το κρυφό επίπεδο δεν είναι αναγκαστικό καθώς υπάρχουν πολλά προβλήματα τα οποία λύνονται χωρίς αυτό, η χρήση του όμως προσδίδει επιπλέον λειτουργικότητα στο δίκτυο.

Αρχικά, λοιπόν, δοκιμάστηκε δίκτυο χωρίς κρυφό επίπεδο. Τα αποτελέσματα που παρέχει το δίκτυο το οποίο αποτελείται μόνο από το επίπεδο εισόδου και το επίπεδο εξόδου, είναι ικανοποιητικά και η συμπεριφορά στις ακραίες τιμές είναι η αναμενόμενη. Πλεονέκτημα σε ένα τέτοιο δίκτυο αποτελεί το μικρό πλήθος συναπτικών βαρών ανάμεσα στους κόμβους του (μόλις 3 συναπτικά βάρη) και οι, αντίστοιχα, λίγες πράξεις μέσα στο δίκτυο - δυο πολλαπλασιασμοί, μία πρόσθεση και μία συνάρτηση ενεργοποίησης. Μειονέκτημά του αποτελεί το μεγάλο πλήθος επαναλήψεων που χρειάζονται κατά τη διαδικασία της εκπαίδευσης ώστε να ρυθμιστούν σωστά τα συναπτικά βάρη του δικτύου.

Η επόμενη δοκιμή έγινε με την προσθήκη κρυφού επιπέδου. Ακολούθησαν δοκιμές με διάφορα πλήθη νευρώνων στο κρυφό επίπεδο σύμφωνα με τους εμπειρικούς κανόνες που περιγράφηκαν στο Κεφάλαιο 2, Παράγραφος 2 και τα αποτελέσματα των δοκιμών παρατίθενται στον Πίνακα 1. Αρχικά το πλήθος νευρώνων του κρυφού επιπέδου τέθηκε ίσο με το πλήθος νευρώνων του επιπέδου εισόδου. Σε αυτή την περίπτωση τα αποτελέσματα δεν είναι ικανοποιητικά καθώς υπήρξαν πολλές περιπτώσεις που το δίκτυο δεν συνέκλινε κατά τη διάρκεια της εκπαίδευσης. Τις φορές που η εκπαίδευση ολοκληρώθηκε και επιτεύχθηκε το επιθυμητό ποσοστό ολικού σφάλματος παρατηρήθηκε ότι το δίκτυο δεν αναγνώριζε ακραίες συνθήκες κατά τη λειτουργία του και παρέδιδε λανθασμένα αποτελέσματα. Για παράδειγμα σε περίπτωση ζεύγους μετρήσεων με τιμές 10.028% για την υγρασία και 29.989°C για τη θερμοκρασία το δίκτυο δίνει πιθανότητα ύπαρξης πυρκαγιάς ίση με 0.82 (82%) ενώ στην πραγματικότητα δεν υπάρχει πυρκαγιά. Αντίστοιχη συμπεριφορά είχε και το δίκτυο με κρυφό επίπεδο που αποτελείται από n+1 νευρώνες, όπου n το πλήθος των νευρώνων εισόδου.

		Train Epochs	TrainError Achieved	ValidationError	Weights Mag	Observations
No Hidden Layer	Iteration 1 Iteration 2 Iteration 3 Iteration 4 Iteration 5	144 166 166 134 165	0.0039 0.0039 0.0039 0.0039 0.0039	213/4066= 5.24% 21/4066=4.94% 205/4066=5.04% 196/4066=4.82% 209/4066=5.14%	6 - 25 6 - 25 6 - 25 6 - 25 6 - 25	Πολλά Epochs σε σχέση με τις υπόλοιπες δομές. Καλό ποσοστό σφάλματος. Μικρά συναπτικά βάρη.
Hidden Layer 2 Neurons	Iteration 1 Iteration 2 Iteration 3 Iteration 4 Iteration 5	5000 5000 214 42 69	- - 0.33 0.0039 0.0038	- - 23/4066=0.56% 121/4066=2.98% 98/4066=2.41%	- - (-187) - 49.000 (-23) - 51 161 - 1108	Το δίκτυο σε αρκετές περιπτώσεις δεν συγκλίνει κατά την εκπαίδευση. Ακόμα και όταν επιτυγχάνεται εκπαίδευση, το δίκτυο αδυνατεί να αναγνωρίσει ακραίες καταστάσεις όπως (10% υγρασία με 29.9°C θερμοκρασία) δίνοντας λανθασμένα αποτελέσματα
Hidden Layer 3 Neurons	Iteration 1 Iteration 2 Iteration 3 Iteration 4 Iteration 5	28 33 5000 21 42	0.0034 0.0036 - 0.0030 0.0039	124/4066=3.04% 121/4066=2.97% - 118/4066=2.90% 132/4066=3.24%	0 - 16 (-97) - 15 - (-24) - 15 (-13) - 152	Το δίκτυο, υπάρχει περίπτωση να μην συγκλίνει. Δυο από τις τέσσερις φορές που υπήρξε επιτυχής εκπαίδευση, το δίκτυο αδυνατούσε να αναγνωρίσει ακραίες περιπτώσεις, δίνοντας λανθασμένα αποτελέσματα
Hidden Layer 4 Neurons	Iteration 1 Iteration 2 Iteration 3 Iteration 4 Iteration 5	54 43 44 45 62	0.0037 0.0027 0.0037 0.0035 0.0037	180/4066=4.42% 72/4066=1.77% 80/4066=1.97% 201/4066=4.94% 197/4066=4.84%	(-243) - 370 (-11) - 35 (-404) - 35 (-455) - 9 (-1304) - 173	Λίγα Epochs κατά την εκπαίδευση, πολλά συναπτικά βάρη με αρκετά μεγάλες τιμές όχι όμως απαγορευτικές. Το ποσοστό σφάλματος είναι ικανοποιητικό και σταθερό.

Πίνακας 1: Τα αποτελέσματα δοκιμών νευρωνικών δικτύων χωρίς και με κρυφό επίπεδο 2, 3 και 4ων νευρώνων.

Αν και κατά τη διάρκεια των δοκιμών πρόεκυψαν δίκτυα με ικανοποιητική συμπεριφορά, το ποσοστό αποτυχίας εκπαίδευσης ή επιτυχίας μη ικανοποιητικής εκπαίδευσης, υπήρξε μεγάλο. Η επόμενη δοκιμή έγινε με κρυφό επίπεδο το οποίο αποτελείτο από 4 νευρώνες. Αρχικά διαπιστώθηκε πως ο μέσος όρος επαναλήψεων που απαιτούντο για να εκπαιδευτεί το δίκτυο ήταν 50, έναντι των 155 επαναλήψεων που απαιτήθηκαν στο δίκτυο χωρίς κρυφό επίπεδο. Το ποσοστό σφάλματος που επιτεύχθηκε ήταν ελαφρώς καλύτερο με μέσο όρο ίσο με 3.588, στις 5 δοκιμαστικές υλοποιήσεις, έναντι του 5.036 που είχαμε στο απλό δίκτυο δυο επιπέδων. Επίσης παρατηρήθηκε ότι το δίκτυο, στις ακραίες περιπτώσεις (τιμές υγρασίας και θερμοκρασίας κάτω από τις τιμές κατωφλίου - 35% και 30°C αντίστοιχα) είναι περισσότερο εκφραστικό από ότι το απλό δίκτυο 2 επιπέδων. Για παράδειγμα στις ακραίες περιπτώσεις όπου η υγρασία κυμαινόταν στο εύρος [35% - 10%] και η θερμοκρασία σε αυτό των [18°C - 29.99°C], το δίκτυο έδινε πιθανότητα ύπαρξης πυρκαγιάς για το ζεύγος τιμών $35\%_{Rhumidity}$ - 29.99°C, ίση με 0.00001739. Η πιθανότητα αυτή αυξανόταν όσο μειωνόταν η υγρασία και αυξανόταν η θερμοκρασία, παίρνοντας μέγιστη τιμή ίση με 27% για το ζεύγος τιμών $10.028\%_{Rhumidity}$ - 29.989°C. Αν και η χρήση του απλού δικτύου δυο επιπέδων δεν είναι απαγορευτική, προτιμήθηκε εδώ η χρήση δικτύου τριών επιπέδων καθώς αυτό εκπαιδεύεται γρηγορότερα από το απλό δίκτυο ενώ παράλληλα το μεγάλο μέγεθος του - συγκριτικά με το μέγεθος του απλού δικτύου των τριών νευρώνων - είναι εύκολα διαχειρίσιμο από τα SPOT, μέσω των οποίων θα υλοποιηθεί.

Η χρήση περισσότερων κόμβων στο κρυφό επίπεδο δοκιμάστηκε δεν έδωσε όμως ικανοποιητικά αποτελέσματα. Από πέντε μέχρι οχτώ κόμβους στο κρυφό επίπεδο, το ποσοστό λάθους στα καινούργια δεδομένα που παρουσιάζονται κατά τη διάρκεια της εκτέλεσης, αυξάνει, ενώ από τους 8 κόμβους και πάνω το δίκτυο αρχίζει να γίνεται δύσκολα διαχειρίσιμο από το Aggregator SPOT, το οποίο το υλοποιεί.

3.4.5. Συναρτήσεις Ενεργοποίησης - Sigmoid / TANH Συναρτήσεις

Αρχικά - όπως αναφέραμε σε προηγούμενο κεφάλαιο - είναι καλό τα δεδομένα ενός νευρωνικού δικτύου να έχουν μέσο όρο ίσο με το 0 και διακύμανση ίση με 1. Επομένως η κανονικοποίηση που έγινε στο υπάρχον Dataset καθώς και στα καινούργια δεδομένα έγινε στο εύρος [-1, 1]. Αυτό είχε ως αποτέλεσμα όλα τα δεδομένα που παρουσιάζονται στο δίκτυο - είτε αυτά είναι μέρος κάποιου Dataset εκπαίδευσης, είτε είναι δεδομένα που παρουσιάζονται στο

δίκτυο κατά τη διάρκεια της εκτέλεσης - να έχουν τιμές από -1 έως 1. Γι αυτό το λόγο χρησιμοποιήθηκε η συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης (TANH) η οποία δέχεται τόσο αρνητικά όσο και θετικά δεδομένα στο εύρος [(-1), 1]. Πλεονέκτημα αυτής της συνάρτησης ενεργοποίησης είναι ότι το δίκτυο εκπαιδεύεται με μόλις 15 epochs (μέσος όρος σε 23 δοκιμαστικές υλοποιήσεις) και τα συναπτικά βάρη που προκύπτουν με τη χρήση της συνάρτησης κυμαίνονται στο εύρος [-7, 7] γεγονός το οποίο τα καθιστά αρκετά μικρότερα από τα αντίστοιχα συναπτικά βάρη που προκύπτουν με τη χρήση της σιγμοειδούς συνάρτησης. Επίσης δεδομένου ότι το εύρος της τιμής εξόδου του δικτύου είναι πλέον το [-1, 1], το δίκτυο είναι περισσότερο εκφραστικό απ ότι στην περίπτωση που το εύρος τιμής εξόδου είναι το [0, 1]. Μειονέκτημα εν προκειμένω αποτελεί το ότι με τη χρήση της συνάρτησης ενεργοποίησης υπερβολικής εφαπτομένης (TANH) αφενός το δίκτυο, χωρίς κρυφό επίπεδο, δεν συγκλίνει και αφετέρου δεν υπάρχει συνοχή ως προς την αποτελεσματικότητα. Κατά τη διαδικασία ελέγχων – δοκιμών προέκυψαν τόσο δίκτυα με μικρά συναπτικά βάρη και μικρό ποσοστό σφάλματος όσο και δίκτυα με μεγάλα συναπτικά βάρη και μεγάλο ποσοστό σφάλματος.

Αντίθετα, η χρήση σιγμοειδούς συνάρτησης ενεργοποίησης, αν και προκαλεί μεγαλύτερα συναπτικά βάρη και απαιτεί περισσότερες επαναλήψεις προκειμένου να επιτευχθεί εκπαίδευση του δικτύου, οδηγεί σε 100% αποτελεσματικά δίκτυα.

Συνοψίζοντας εάν το επιθυμητό σενάριο είναι να εκπαιδεύεται εκ νέου ένα νευρωνικό δίκτυο για κάθε Aggregator που τοποθετείται στο δίκτυο του συστήματος τότε η ασφαλέστερη επιλογή είναι η σιγμοειδής συνάρτηση ενεργοποίησης. Αντίθετα εάν δεν απαιτείται εκ νέου εκπαίδευση νευρωνικού δικτύου για κάθε Aggregator SPOT που τοποθετείται στο δίκτυο και αρκεί να υπάρχει ένα μοντέλο το οποίο αποστέλλεται σε κάθε Aggregator, τότε μπορούμε να εκπαιδεύουμε νευρωνικά δίκτυα μέχρι να επιτύχουμε ένα με τα επιθυμητά χαρακτηριστικά (μικρά συναπτικά βάρη και μικρό ποσοστό σφάλματος), να αποθηκεύσουμε τα συναπτικά του βάρη και στη συνέχεια να αποστέλλουμε αυτά τα βάρη σε οποίο Aggregator τοποθετηθεί στο δίκτυο του συστήματος. Για παράδειγμα, συναπτικά βάρη που συνιστούν ένα τέτοιο δίκτυο - με συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης, μικρό ποσοστό σφάλματος και μικρά συναπτικά βάρη - είναι τα εξής :

[0.15184915612206612 1.1278495349135578 0.799649049324257 -0.3156726183737362 6.185441207681171 1.272746805465911 0.7013867361897543 -5.861699349730314 -0.31417107844460157 0.388679749552616 1.3365789073042138 0.30732244373244083 2.5813366518955325 1.3022169935732792 0.13428641325680904 -2.696802521813268 0.05228459678368967].

Εάν αυτό το διάνυσμα συναπτικών βαρών, φορτωθεί σε οποιοδήποτε δίκτυο υλοποιούμενο από κάποιο Aggregator, μπορούμε να είμαστε σίγουροι ότι θα λειτουργήσει ικανοποιητικά. Στην παρούσα εργασία προτιμήθηκε η σιγμοειδής συνάρτηση ενεργοποίησης καθώς καμία τιμή υγρασίας ή θερμοκρασίας, από τις υπάρχουσες δεν είναι αρνητική, οπότε μοιάζει περιττό να κανονικοποιήσει κανείς θετικές τιμές σε ένα εύρος αρνητικών – θετικών τιμών. Επίσης επειδή η ζητούμενη έξοδος του νευρωνικού δικτύου είναι η πιθανότητα ύπαρξης φωτιάς, θεωρήθηκε προτιμότερη η σιγμοειδής συνάρτηση η οποία έχει εύρος [0, 1] και μπορεί να θεωρηθεί απευθείας ως πιθανότητα χωρίς περαιτέρω επεξεργασία [15] ενώ για τη συνάρτηση ενεργοποίησης υπερβολικής εφαπτομένης πρέπει να γίνει αντιστοίχηση του εύρους [-1, 1] στο εύρος [0,1].

3.4.6. Reservoir Sampling & Moving Average

Για τη δειγματοληψία των μετρήσεων, από την πλευρά του Aggregator, χρησιμοποιήθηκαν και συγκριθήκαν δυο αλγόριθμοι. Πρώτος χρησιμοποιήθηκε ο αλγόριθμος Reservoir Sampling και στη συνέχεια κρίθηκε απαραίτητο να δοκιμαστεί ο αλγόριθμος Moving Average. Και οι δυο αλγόριθμοι δοκιμάστηκαν ως μέσο λύσης του προβλήματος της σύγκρουσης πακέτων - που συμβαίνει όταν συσκευές στέλνουν δεδομένα ταυτόχρονα στον ίδιο παραλήπτη - και του προβλήματος των ακραίων στιγμιαίων μετρήσεων.

Reservoir Sampling

Το Reservoir Sampling [16] [17] αποτελεί μία οικογένεια τυχαιοποιημένων αλγορίθμων για τυχαία επιλογή k στοιχείων από λίστα n στοιχείων, όπου το n είναι είτε πολύ μεγάλος είτε άγνωστος αριθμός. Συνήθως το n είναι τόσο μεγάλο ώστε να μην χωράει στην κύρια μνήμη του συστήματος (την Ram ενός υπολογιστή ή την flash μνήμη ενός μικροελεγκτή). Ιδιαιτερότητα των αλγορίθμων αυτών αποτελεί το ότι δεν περιορίζονται στην τυχαία επιλογή ενός πλήθους k στοιχείων από μία λίστα μεγέθους n αλλά παράλληλα εξασφαλίζουν ότι το πλήθος k που επιλέγεται, είναι ομοιόμορφα κατανεμημένο στη λίστα.

Για την υλοποίηση του αλγορίθμου Reservoir Sampling απαιτούνται τα ακόλουθα βήματα:

1. Δημιουργία ενός πίνακα reservoir k θέσεων: reservoir[k] (το μέγεθος του πίνακα – στην περίπτωσή μας ίσο με 5 - ορίζεται από το σχεδιαστή. Θα μπορούσε να επιλεγεί

οποιαδήποτε τιμή με την προϋπόθεση ότι το μέγεθος της μπορεί να υποστηριχτεί από την πλατφόρμα στην οποία εκτελείται ο αλγόριθμος.

2. Αντιγραφή των k πρώτων στοιχείων από τη ροη δεδομένων.
3. Μετά την πλήρωση του reservoir ορίζεται μία μεταβλητή πιθανότητας αποδοχής: acceptance probability = reservoir.size/streamIndex, Η μεταβλητή αυτή ουσιαστικά εκφράζει την πιθανότητα που έχει μία τρέχουσα μέτρηση να αντικαταστήσει κάποια από τις μετρήσεις που υπάρχουν στο reservoir (πλήθους reservoir.size()).
4. Ορισμός μίας μεταβλητής τυχαίας πιθανότητας με χρήση της συνάρτησης:

   Random():ranProp=newRandom();

   ranProp.setSeed(System.currentTimeMillis());

   randomPropability=(double)ranProp.nextDouble();
5. Εάν η τυχαία πιθανότητα randomPropability είναι μικρότερη της πιθανότητας αποδοχής της τρέχουσας μέτρησης τότε δημιουργείται ένας τυχαίος αριθμός από το 0 έως το reservoir.size() - 1 και στη θέση του reservoir που υποδεικνύεται από αυτόν τον αριθμό, εισάγεται η τρέχουσα μέτρηση, αντικαθιστώντας αυτή που υπήρχε ήδη εκεί.

Ο κώδικας που υλοποιεί τον αλγόριθμο Reservoir Sampling είναι ο ακόλουθος:

```
public void sample (LLNode currentSpot, double humidityRead,
   double temperatureRead, int streamIndex) {

    double acceptancePropability = (double)ReservoirSize/streamIndex;
     if (streamIndex < ReservoirSize){
          currentSpot.humidityData.addElement(Double.toString(humidityRead));
          currentSpot.temperatureData.addElement(Double.toString(temperatureRead));
}
     else {
          ranProp.setSeed(System.currentTimeMillis());
          randomPropability=(double)ranProp.nextDouble();

          if (randomPropability < acceptancePropability){
                    ranInd.setSeed(System.currentTimeMillis());
                    randomIndex=(int)ranInd.nextInt((ReservoirSize));
                    currentSpot.humidityData.setElementAt(Double.toString(humidityRead),randomIndex);
                    currentSpot.temperatureData.setElementAt(Double.toString(temperatureRead),randomIndex);
          }
     }
}
```

Προκειμένου να γίνει κατανοητός ο τρόπος λειτουργίας του υπόψη αλγόριθμου ας υποθέσουμε, στην απλούστερη περίπτωση, ότι θέλουμε να αποθηκεύσουμε ένα τυχαίο δείγμα, ομοιόμορφα κατανεμημένο, από μία λίστα 5 στοιχείων σε ένα διάνυσμα (reservoir) 5 θέσεων. Σύμφωνα με το Βήμα 1., πιο πάνω, αφού η λίστα και το διάνυσμα έχουν το ίδιο μέγεθος, τότε στο τέλος του αλγορίθμου αντιγράφονται όλα τα στοιχεία της λίστας στο reservoir.

Ας υποθέσουμε τώρα ότι έχουμε 10 στοιχεία στη λίστα και ένα reservoir 5 θέσεων. Σύμφωνα με το Βήμα 1, στο reservoir έχουμε τα 5 πρώτα στοιχεία της λίστας. Έρχεται τώρα το 6ο στοιχείο από τη λίστα. Η πιθανότητα να επιλεχθεί για αποθήκευση το 6ο στοιχείο είναι ίση με $P_{chooseCurrent} = \frac{k}{i} = \frac{5}{6}$, όπου k είναι το μέγεθος του reservoir, και i είναι το streamIndex δηλαδή η σειρά με την οποία ήρθε η τρέχουσα μέτρηση. Για κάθε στοιχείο που είναι ήδη στο reservoir, η πιθανότητα να παραμένει στο reservoir τη στιγμή που έρχεται η τρέχουσα μέτρηση είναι ίση με $P_{alreadyIn} = \frac{k}{i-1} = \frac{5}{6-1} = 1$. Η πιθανότητα για κάθε στοιχείο να αντικατασταθεί από την τρέχουσα μέτρηση είναι ίση με την πιθανότητα να επιλεγεί για αποθήκευση η τρέχουσα μέτρηση επί την πιθανότητα να επιλεγεί το συγκεκριμένο στοιχείο $P_{chooseStored}$ (η οποία είναι πάντα ίση με $\frac{1}{\kappa}$), άρα $P_{replace} = P_{chooseCurrent} * P_{chooseStored} = \frac{k}{i} * \frac{1}{k} = \frac{1}{i} = \frac{1}{6}$ και η πιθανότητα για κάθε στοιχείο του reservoir να μην επιλεγεί προς αντικατάσταση είναι ίση με $P_{survive} = 1 - P_{replace} = 1 - \frac{1}{i} = \frac{i-1}{i} = \frac{5}{6}$. Επομένως η συνολική πιθανότητα ενός αντικειμένου να συνεχίσει να είναι στο reservoir μετά από i μετρήσεις είναι ίση με την πιθανότητα το στοιχείο αυτό να είναι ήδη στο reservoir επί την πιθανότητα να μην επιλεγεί για αντικατάσταση από την $i_{ωστη}$ μέτρηση. Άρα $P_{final} = P_{alreadyIn} * P_{survive} = \frac{k}{i-1} * \frac{i-1}{i} = \frac{k}{i} = \frac{5}{6}$.

Αντίστοιχα βλέπουμε πως για το 7ο στοιχείο ισχύουν τα εξής:

- $P_{chooseCurrent} = \frac{k}{i} = \frac{5}{7}$
- $P_{alreadyIn} = \frac{k}{i-1} = \frac{5}{7-1} = \frac{5}{6}$
- $P_{chooseStored} = \frac{1}{\kappa} = \frac{1}{5}$
- $P_{replace} = P_{chooseCurrent} * P_{chooseStored} = \frac{k}{i} * \frac{1}{k} = \frac{1}{i} = \frac{1}{7}$
- $P_{survive} = 1 - P_{replace} = 1 - \frac{1}{i} = \frac{i-1}{i} = \frac{6}{7}$.
- $P_{final} = P_{alreadyIn} * P_{survive} = \frac{k}{i-1} * \frac{i-1}{i} = \frac{k}{i} = \frac{5}{7}$

Βλέπουμε δηλαδή πως όλα τα στοιχεία έχουν την ίδια πιθανότητα να παραμένουν στο reservoir μετά την ολοκλήρωση των πράξεων ($P_{chooseCurrent} = P_{final}$). Το πρόβλημα συνεπώς συνίσταται στο ότι αν για ένα συγκεκριμένο πλήθος μετρήσεων όλες οι μετρήσεις έχουν την ίδια πιθανότητα να είναι μέσα στο reservoir τότε όσο το πλήθος των μετρήσεων αυξάνει τόσο μικραίνει η πιθανότητα κάποια από τις τρέχουσες μετρήσεις να αποθηκευθεί. Συγκεκριμένα είδαμε πως η πιθανότητα να αποθηκευθεί η εβδόμη μέτρηση και η πιθανότητα να παραμείνει κάποια από τις ήδη υπάρχουσες μετρήσεις στο reservoir, είναι και οι δυο ίσες με $\frac{5}{7}$. Αντίστοιχα η πιθανότητα να αποθηκευθεί η εκατοστή μέτρηση και η πιθανότητα να επιβιώσει κάποια από τις ήδη υπάρχουσες στο reservoir μετρήσεις είναι ίσες με $\frac{5}{100}$.

Δηλαδή αν και οι αποθηκευμένες και η τρέχουσα μέτρηση έχουν ίση πιθανότητα να επιβιώσουν για έναν συγκεκριμένο γύρο, οι μετρήσεις των επόμενων γύρων έχουν μικρότερες πιθανότητες να είναι στο reservoir από αυτές των μετρήσεων προηγούμενων γύρων. Πρακτικά - στο σύστημα που υλοποιείται στην παρούσα εργασία - αυτό σημαίνει πως όσο περνάει η ώρα, οι μετρήσεις του reservoir θα ανανεώνονται όλο και λιγότερο με αποτέλεσμα από μία συγκεκριμένη χρονική στιγμή και πέρα να μην είναι πλέον αντιπροσωπευτικές των περιβαλλοντικών συνθηκών που επικρατούν. Στη δοκιμή που έγινε στο παρόν σύστημα παρατηρήθηκε ότι μετά από περίπου 18.000 μετρήσεις (2,5 ώρες λειτουργίας με συχνότητα δειγματοληψίας τα 500 ms) η επόμενη μέτρηση που αποθηκεύθηκε ήταν η μέτρηση με αύξοντα αριθμό 22.345. Περίπου 40 λεπτά μετρήσεων (4.435 μετρήσεις) δεν αποθηκεύθηκαν και οτιδήποτε συνέβη κατά τη διάρκεια αυτών των μετρήσεων πέρασε απαρατήρητο από το σύστημα. Γι αυτό το λόγο η μέθοδος reservoir sampling κρίθηκε ακατάλληλη και χρησιμοποιήθηκε η μέθοδος κινούμενου μέσου (moving average)

Moving Average

Ο κινούμενος μέσος [18][19][20] είναι ένα εργαλείο που κατά κύριο λόγο χρησιμοποιείται στη στατιστική ανάλυση προκειμένου να αναλύουμε δεδομένα δημιουργώντας μία σειρά από μέσους διαφορετικών υποσυνόλων ενός υπερσυνόλου. Ας υποθέσουμε ότι έχουμε μία σειρά από δεδομένα και ένα υποσύνολο γνωστού μεγέθους (ας υποθέσουμε ίσο με 5) το πρώτο στοιχείο του κινούμενου μέσου αποκτάται υπολογίζοντας τον μέσο των 5 πρώτων στοιχείων της σειράς δεδομένων. Στη συνέχεια, το υποσύνολο των 5 στοιχείων μεταβάλλεται ολισθαίνοντας κατά μία

θέση "μπροστά" αφαιρώντας δηλαδή την πρώτη τιμή από τις 5 και προσθέτοντας την έκτη τιμή της σειράς δεδομένων. Η διαδικασία αυτή δημιουργεί ένα καινούργιο υποσύνολο 5 στοιχείων του οποίου υπολογίζουμε και πάλι τον μέσο όρο. Αυτή η διαδικασία επαναλαμβάνεται μέχρι να χρησιμοποιηθεί το σύνολο των δεδομένων (από τη σειρά δεδομένων). Στην παρούσα εργασία δεδομένα αποτελούν οι μετρήσεις που αποστέλλουν διαρκώς τα Sampler SPOT και συνεπώς η συγκεκριμένη διαδικασία δεν τερματίζεται.

Από απόψεως μαθηματικών ο κινούμενος μέσος είναι ένα είδος συνέλιξης και μπορεί να θεωρηθεί αντίστοιχος ενός βαθυπερατού φίλτρου που χρησιμοποιείται σε επεξεργασία σήματος. Η αναγνώριση τάσεων είναι ένα από τα χαρακτηριστικά του κινούμενου μέσου, όμως πρέπει να διευκρινιστεί ότι δεν μπορεί να προβλέψει μία τάση (ανοδική η καθοδική) παρά μόνο να την επιβεβαιώσει αφού η ίδια η τάση καθιερωθεί. Συναπτικά μία τάση μπορεί να χαρακτηριστεί ως ανοδική ή καθοδική όταν η τρέχουσα τιμή δεδομένων είναι υψηλότερη ή χαμηλότερη της τιμής του κινούμενου μέσου όρου αντιστοίχως. .

Στο σύστημα που υλοποιήθηκε για την παρούσα εργασία, χρησιμοποιήθηκε μία παραλλαγή του κινούμενου μέσου ορού - ο σταθμισμένος κινούμενος μέσος (WMA). Εν προκειμένω αντί να προσθέτουμε τα δεδομένα p_i ενός υποσυνόλου και να διαιρούμε με το πλήθος n των δεδομένων (*Εξίσωση* 1), όπως στην περίπτωση του απλού κινούμενου μέσου (SMA), προσθέτουμε τα δεδομένα, αφού προηγουμένως πολλαπλασιάσουμε κάθε ένα από αυτά ένα βάρος n που εκφράζει το ποσοστό επιρροής που έχει το συγκεκριμένο δεδομένο στην έξοδο, και διαιρούμε με το άθροισμα των βαρών (*Εξίσωση* 3).

$$SMA = \frac{p_M + p_{M-1} + \ldots + p_{M-(n-1)}}{n} \qquad \textit{Εξίσωση 1}$$

$$SMA_{current} = SMA_{previous} + \frac{p_M}{n} - \frac{p_{M-n}}{n} \qquad \textit{Εξίσωση 2}$$

$$WMA_M = \frac{np_M + (n-1)p_{M-1} + \ldots + 2p_{(M-n-2)} + \ldots + p_{(M-n-1)}}{n + (n-1) + \cdots + 2 + 1} \qquad \textit{Εξίσωση 3}$$

Στην περίπτωση του SMA, παρόλο που κάθε κινούμενος μέσος όρος είναι ο μέσος όρος ενός διαφορετικού υποσυνόλου ενός υπερσυνόλου, δεν χρειάζεται να υπολογίζουμε εκ νέου το καινούργιο άθροισμα των στοιχείων και να διαιρούμε με το πλήθος των μετρήσεων κάθε φορά που έχουμε καινούργιο υποσύνολο. Όπως φαίνεται στη συνάρτηση 2, ο τρέχων SMA είναι ίσος με το άθροισμα του παλιού SMA με το πηλίκο του καινούργιου δεδομένου δια του πλήθους των μετρήσεων, μείον το πηλίκο του δεδομένου που μόλις αφαιρέθηκε από το υποσύνολο δια του

πλήθους των δεδομένων. Αντίστοιχα οι *Εξισώσεις* 4, 5 και 6, κατωτέρω, δείχνουν πως μπορεί να υπολογιστεί ο καινούργιος WMA χωρίς να υπολογιστεί εκ νέου το πηλίκο του αθροίσματος των γινομένων των δεδομένων επί τα βάρη τους δια του αθροίσματος των βαρών. Συγκεκριμένα η διαφορά του αριθμητή του W_{M+1} με τον αριθμητή του W_M ειναι ίση με $np_{M+1} - p_M - \ldots - p_{(M-n+1)}$. Εάν ονομάσουμε $Total_M$ αυτή τη διάφορα τότε έχουμε:

$$Total_{M+1} = Total_M + p_{M+1} - p_{M-n+1} \qquad \textit{Εξίσωση 4}$$

$$Numerator_{M+1} = Numerator_M + np_{M+1} - Total_M \qquad \textit{Εξίσωση 5}$$

$$WMA_{M+1} = \frac{Numerator_{M+1}}{n+(n-1)+\cdots+2+1} \qquad \textit{Εξίσωση 6}$$

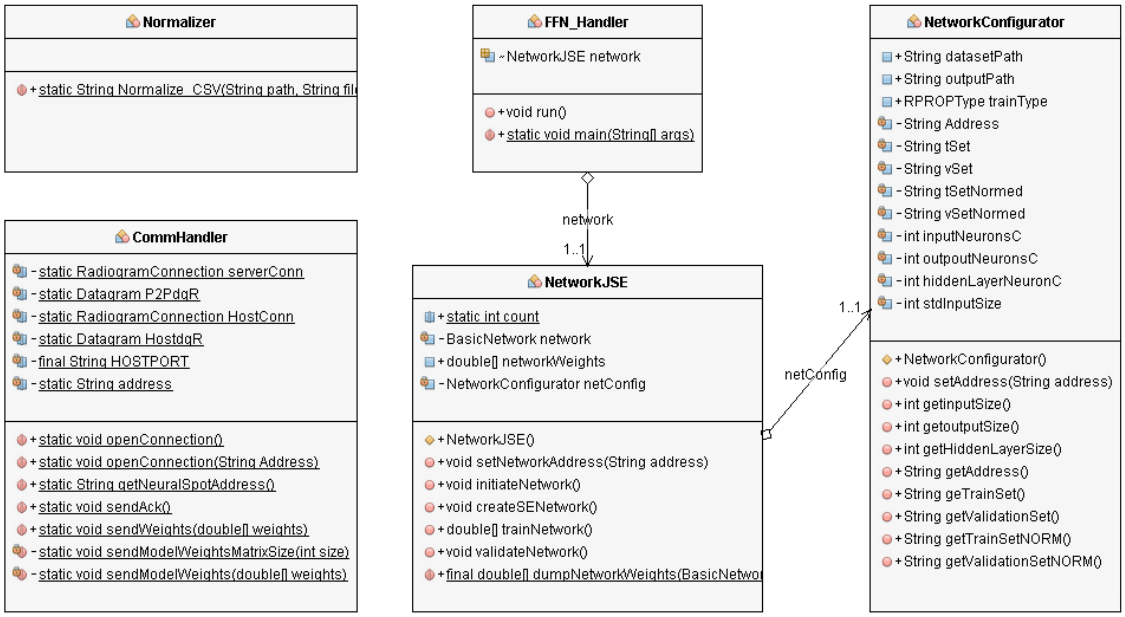
Εδώ πρέπει να σημειωθεί πως στο σύστημα που υλοποιήθηκε για την παρούσα εργασία, οι τιμές του σταθμισμένου κινούμενου μέσου υπολογίζονται κάθε φορά εκ νέου και αυτό γιατί το πλήθος των μετρήσεων που αποτελούν το υποσύνολο από το οποίο υπολογίζεται κάθε φορά ο WMA, είναι τέτοιο σε μέγεθος (ίσο με 5) που οι πράξεις που γίνονται για τον εκ νέου υπολογισμό είναι λιγότερες από τις πράξεις που θα πραγματοποιούσαμε εάν υπολογίζαμε το WMA με τον τρόπο που υποδεικνύεται από τις Συναρτήσεις 4,5 και 6.

4. Midlets & Host Application. Κλάσεις και μέθοδοι.

4.1. Host Application FFN_Handler.

Host Application FFN_Handler είναι η host εφαρμογή που εκτελείται στον υπολογιστή με στόχο την αποστολή στο Aggregator SPOT των συναπτικών βαρών του δικτύου που αυτό θα υλοποιεί.. Προκειμένου να επιτευχθεί ο συγκεκριμένος στόχος η εφαρμογή δημιουργεί ένα νευρωνικό δίκτυο όμοιο με αυτό του Aggregator SPOT, το εκπαιδεύει και αφού ολοκληρώσει επιτυχώς την εκπαίδευση αποστέλλει τα συναπτικά βάρη που προκύπτουν από την εκπαίδευση, στο συγκεκριμένο Aggregator SPOT. Η διαδικασία ολοκληρώνεται σε επιμέρους φάσεις στις οποίες χρησιμοποιούνται οι παρακάτω κλάσεις.

1. Η κλάση FFN_Handler. Αποτελεί την κύρια κλάση της εφαρμογής. Η κλάση αυτή δεν έχει κάποια δομική λειτουργία αλλά χρησιμοποιεί άλλες κλάσεις για να εκτελέσει τις απαραίτητες ενέργειες.
2. NetworkJSE. Η κλάση αυτή καλείται από την FFN_Handler για να δημιουργήσει, χρησιμοποιώντας τη βιβλιοθήκη Encog, ένα νευρωνικό δίκτυο, να το εκπαιδεύσει, να το επαληθεύσει και τέλος να κωδικοποιήσει το δίκτυο σε ένα διάνυσμα από double αριθμούς - τα συναπτικά βάρη.
3. NetworkConfigurator. Η κλάση αυτή περιέχει πληροφορίες για το νευρωνικό δίκτυο που θα δημιουργηθεί, όπως το μονοπάτι για τα Dataset εκπαίδευσης/επαλήθευσης τα ονόματα των αρχείων και τα μεγέθη των επιπέδων του νευρωνικού δικτύου. Χρησιμοποιείται από την κλάση NetworkJSE.
4. CommHandler. Είναι η κλάση που χειρίζεται όλες τις επικοινωνίες ανάμεσα στην Host εφαρμογή και οποιοδήποτε Aggregator SPOT τοποθετηθεί στο δίκτυο. Είναι στατική κλάση και δεν είναι απαραίτητο να δημιουργηθεί αντικείμενο της κλάσης αυτής. Καλείται μόνο από την κλάση FFN_Handler.
5. Normalizer. Η κλάση αυτή χρησιμοποιεί έτοιμες κλάσεις του Encog ώστε να κανονικοποιήσει τα Dataset εκπαίδευσης και επαλήθευσης που χρησιμοποιούνται από το νευρωνικό δίκτυο. Είναι στατική κλάση και τη χρησιμοποιεί μόνο η κλάση NetworkJSE.



Εικόνα 30. Class διαγράμματα των κλάσεων της host εφαρμογής.

4.1.1 FFN_Handler.class

```
public class FFN_Handler {

    NetworkJSE network = new NetworkJSE();

    public void run() throws UnsupportedEncodingException,
FileNotFoundException{

        long ourAddr = RadioFactory.getRadioPolicyManager().getIEEEAddress();
        System.out.println("Our radio address = " +
IEEEAddress.toDottedHex(ourAddr));

        while (true){

            network.setNetworkAddress(CommHandler.getNeuralSpotAddress());
            network.initiateNetwork();
            network.validateNetwork();
            CommHandler.sendWeights(network.networkWeights);
        }
    }
    /**
     * Start up the host application.
     *
     * @param args any command line arguments
     * @throws java.io.UnsupportedEncodingException
     * @throws java.io.FileNotFoundException
     */
    public static void main(String[] args) throws
UnsupportedEncodingException, FileNotFoundException {
        OTACommandServer.start("FFN_Handler");

        FFN_Handler app = new FFN_Handler();
        app.run();
    }
}
```

Εικόνα 31. Ο κώδικας της κλάσης FFN_Handler

4.1.2NetworkConfigurator.class

```
package org.sunspotworld;

import org.encog.neural.networks.training.propagation.resilient.RPROPType;

/**
 * A Class containing information for the neural network to be created/trained.
 * @author Michalis
 */
public class NetworkConfigurator {

    /**
     * The path to the training Dataset. Path is absolute and must be changed
     * accordingly when project run on another computer.
     */
```

```
    public final String datasetPath="C:\\Users\\Michalis\\Documents\\SUNSpot_Apps\\
                                     FFN_Handler\\dataSets";

    /**
     * The location where app stores neural network validation results.
     * A different file created for each of the 4 possible trainType.
     * Path is absolute and must be changed accordingly when project run
     * on another computer.
     */
    public final String outputPath ="C:\\Users\\Michalis\\Documents\\SUNSpot_Apps\\
                                     FFN_Handler\\neuralResults";
    /**
     * Train type for the resilient propagation training method.
     */
    public final RPROPType trainType = RPROPType.iRPROPp;

    /**
     * Current SPOT address.
     */
    private String Address;

    /**
     * The training set that will be used for the current Network.
     */
    private String tSet;

    /**
     * The validation set that will be used for the current Network.
     */
    private String vSet;

    /**
     * The normalized training set that will be used for the current Network.
     */
    private String tSetNormed;

    /**
     * The normalized validation set that will be used for the current Network.
     */
    private String vSetNormed;
    /**
     * Input level neuron count.
     */
    private  int inputNeuronsC;

    /**
     * Output level neuron count.
     */
    private  int outpoutNeuronsC;

    /**
     * Hidden layer neuron count.
     */
    private int hiddenLayerNeuronC = 4;

    /**
     * Networks standard input level Neuron count.
     */
    private final int stdInputSize = 2;

    /**
     * Class Constructor. Initializes members and variables.
     */
```

```
    public NetworkConfigurator(){}

    /**
     * Setter Method.
     * @param address Stores the current SPOT address
     */
    public final void setAddress(String address){}

    /**
     * Getter Method.
     * @return networks input level neuron count.
     */
    public final int getinputSize(){}

    /**
     * Getter Method.
     * @return networks output level neuron count.
     */
    public final int getoutputSize(){}

    /**
     * Getter Method.
     * @return networks hidden layer neuron count.
     */
    public final int getHiddenLayerSize(){}

    /**
     * Getter Method.
     * @return current SPOT's address.
     */
    public final String getAddress(){}

    /**
     * Getter Method.
     * @return a String indicating the training set to be used
     * for network training.
     */
    public final String geTrainSet(){}

    /**
     * Getter Method.
     * @return a String indicating the validation set to be used
     * for network validation.
     */
    public final String getValidationSet(){}

    /**
     * Getter Method.
     * @return a String indicating the normalized training set
     * to be used for network validation.
     */
    public final String getTrainSetNORM(){}

    /**
     * Getter Method.
     * @return a String indicating the normalized validation set
     * to be used for network validation.
     */
    public final String getValidationSetNORM(){}
}
```

4.1.3. CommHandler.class

```
/**
 * Class responsible for all communication between host app and Aggregator SPOT.
 * Static class needs not to be instantiated.
 */
public class CommHandler {

    /**
     * Opens a general purpose connection. All Aggregator SPOT will send
     * weight requests via this connection.
     */
    public static void openConnection(){}

    /**
     * Opens a connection with a specific Aggregator SPOT.
     * @param Address a String representing the Aggregator SPOT address.
     */
    public static void openConnection(String Address){}

    /**
     * Halts until an Aggregator SPOT sends a message. Once message received
     * Aggregator SPOT address extracted. GP connection closes and dedicated
     * connection between host app - specific spot established.
     * @return
     */
    public static String getAggregatorSpotAddress(){}

    /**
     * Acknowledgement message. Host App confirms that SPOT address received.
     */
    public static void sendAck(){}

    /**
     * Opens previously dedicated connection sends the weights array length for
     * validation and then sends the actual synaptic weights.
     * @param weights. array holding the synaptic weights of the trained
     * neural network.
     */
    public static void sendWeights(double[] weights){}

    /**
     * The method that sends the synaptic weights Matrix size to the SPOT.
     * @param size the size of the weight matrix.
     */
    private static void sendModelWeightsMatrixSize(int size){}

    /**
     * Method that sends the actual weight array.
     * @param weights Neural Networks synaptic weights
     */
    private static void sendModelWeights(double[] weights){}
}
```

4.1.4. NetworkJSE

```
/**
 * Class that creates - trains - validates the neural network and then
 * extracts the synaptic weights of the trained/validated network to an
 * array of doubles [double[]]. Note that there is only one network
 * created. Each time an Aggregator SPOT requests synaptic weights the
 * network resets and trains again.
 * @author Michalis
 */
public class NetworkJSE {

    /**
     * Class Constructor. Initiates network weights. Instantiates
     * a new NetworkConfigurator and creates the neural network to
     * be trained.
     */
    public NetworkJSE (){}

    /**
     * Setter Method.
     * @param address. Sets the networkConfigurator address
     * equal to current SPOT address.
     */
    public void setNetworkAddress(String address){}

    /**
     * Resets the network created by Constructor. Normalizes training
     * and validation DataSets and then calls method that trains network.
     */
    public void initiateNetwork(){}

    /**
     * Create a encogSE NNModel.
    */
    public final void createSENetwork(){}

    /**
     * Trains the neural network up to the desired error rate. If error
     * rate is still higher than 0.01 after 3000 Dataset iterations then
     * training gets aborted, networks synaptic weights get randomized
     * to an adjustable span and training restarts.
     * @return an array of doubles containing the networks synaptic
     * weights
     */
    public final double[] trainNetwork() {}

    /**
     * Validates the trained neural network. Prints results to .txt file
     * and IDE output window.
     * @throws FileNotFoundException
     * @throws UnsupportedEncodingException
     */
    public final void validateNetwork() throws FileNotFoundException,
            UnsupportedEncodingException  {}

    /**
     * Encodes the network's synaptic weights in to an array of doubles.
     * @param network. The network needed to be encoded to an array
     * @return. The double[] containing the weights.
     */
    public static final double[] dumpNetworkWeights(BasicNetwork network){}
}
```

4.1.5. Normalizer

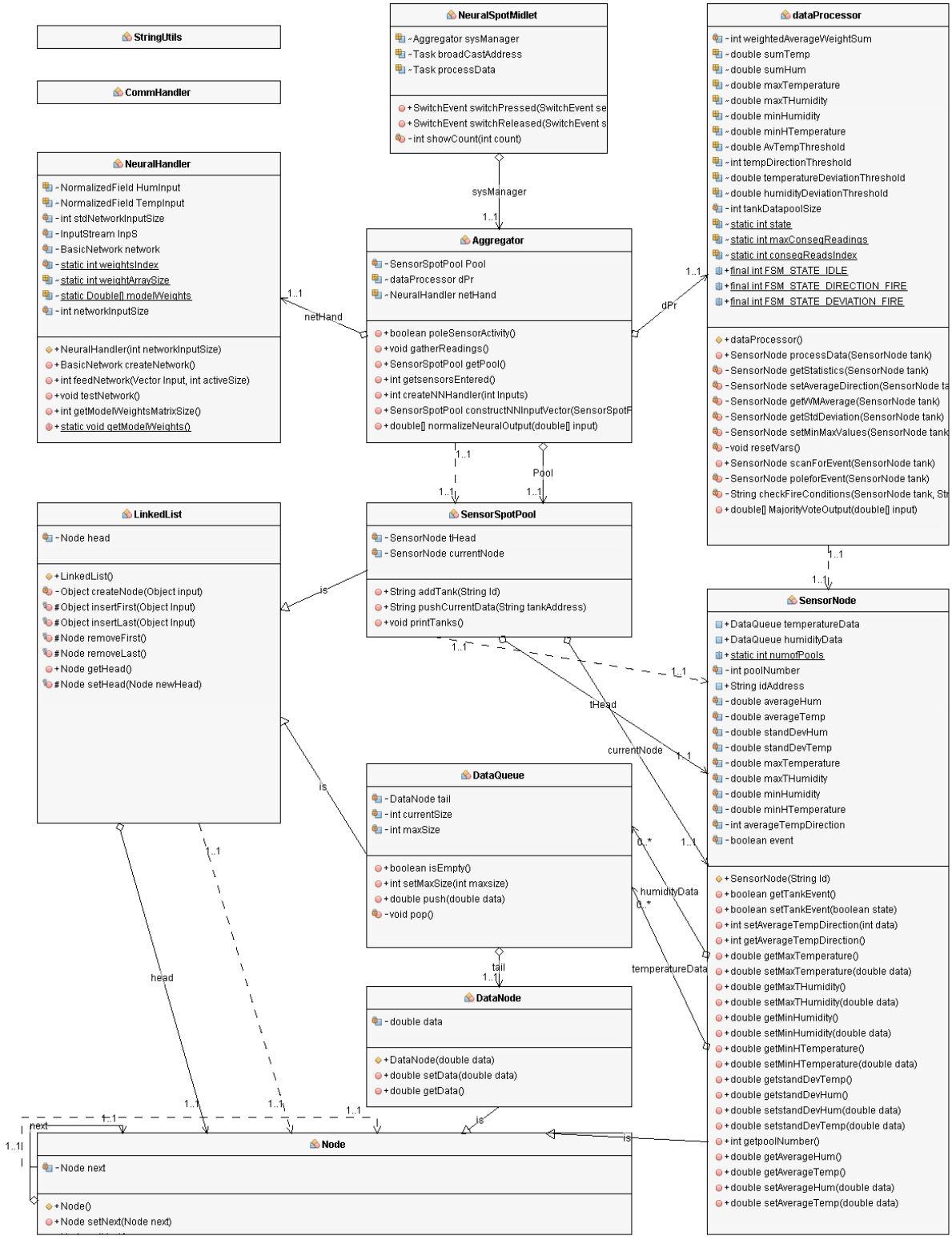
```
/**
 * Class that contains the tool needed to normalize tha .CSV Datasets.
 * Uses encog classes.
 * @author Michalis
 */
public final class Normalizer {

    /**
     * Normalizes the Datasets to a desired range.
     * @param path. The pat to the Datasets.
     * @param filenameINP. the name of the dataset to be used.
     * @param filenameOUTP. The output of the normalization i.e the normalized
     *                      dataset.
     * @param type. The count of input layer neurons. Was used when multiple
     *              input architecture used.
     * @return a string holding the name of the output file.
     */
    public static String Normalize_CSV (String path, String filenameINP,
                                                String filenameOUTP,int type){}
}
```

4.2 Aggregator Midlet

Είναι το midlet που εκτελείται στα Aggregator SPOT. Εκτελεί δυο βασικές λειτουργίες - υλοποιεί το νευρωνικό δίκτυο του οποίου τα βάρη απέκτησε από την host εφαρμογή και τροφοδοτεί το δίκτυο, με μετρήσεις που λαμβάνει από Sampler SPOT. Τα βήματα που ακολουθούντα προκειμένου να εκτελέσει τις δυο αυτές λειτουργίες είναι - με χρονική σειρά τα εξής:

1. Ανοίγει μία σύνδεση τύπου Broadcast και εκπέμπει μήνυμα που περιέχει μόνο τη διεύθυνση του (ουσιαστικά αυτό είναι το μήνυμα αίτησης συναπτικών βαρών) μέχρι να τη λάβει η host εφαρμογή. Παράλληλα ανοίγει άλλη μία σύνδεση και σε αυτή τη σύνδεση περιμένει μήνυμα Αναγνώρισης (ACK) από την host εφαρμογή.
2. Μόλις λάβει μήνυμα αναγνώρισης κλείνει την Broadcast σύνδεση και ανοίγει αφοσιωμένη σύνδεση με την host εφαρμογή. Δημιουργεί ένα νευρωνικό δίκτυο και αφού λάβει τα συναπτικά βάρη από την host εφαρμογή, τα φορτώνει στο δίκτυο και το επαληθεύει. Τα αποτελέσματα της επαλήθευσης τυπώνονται στο παράθυρο εξόδου του SPOT.
3. Κλείνει τη σύνδεση με την host εφαρμογή και ανοίγει μία σύνδεση στο port 82, στην οποία θα αποστέλλουν μετρήσεις τα Sampler SPOT. Στη συνέχεια αναμένει πλέον μήνυμα από κάποιο Sampler SPOT.
4. Μόλις λάβει μήνυμα από κάποιο Sampler SPOT, είτε εισάγει το SPOT στη λίστα με τα γνωστά του Sampler SPOT, εάν είναι η πρώτη φορά που το συγκεκριμένο Sampler αποστέλλει μήνυμα στο Aggregator, είτε αποθηκεύει τις μετρήσεις του Sampler.
5. Ενεργοποιεί ένα task, το οποίο κάθε κάποιο χρονικό διάστημα επεξεργάζεται τα δεδομένα που έχει μαζέψει από τα Sampler SPOT.
6. Εάν οι μετρήσεις ξεπερνούν κάποιες στατιστικές τιμές κατωφλίου τότε οι μετρήσεις τροφοδοτούνται στο νευρωνικό δίκτυο
7. Η έξοδος του νευρωνικού δικτύου τυπώνεται στο παράθυρο εξόδου του Aggregator SPOT.

Όπως φαίνεται και στην Εικόνα 32, η διαδικασία που περιγράφετεαι παραπάνω, τμηματοποιήθηκε και οι επιμέρους διαδικασίες κατανεμήθηκαν σε 13 συνολικά κλάσεις.



Εικόνα 32. Οι κλάσεις του Aggregator Midlet και οι σχέσεις μεταξύ τους, σε διάγραμμα κλάσεων UML.

4.2.1. Aggregator.class

```
protected void startApp() throws MIDletStateChangeException {

    sw1.addISwitchListener(this);
    sw2.addISwitchListener(this);
    leds.setRGB(0, 0, 255);
    boolean SPOTKnown = false;

    processData = new Task(1*1000) {
        public void doTask() throws Exception {

            sysManager.checkForFire();
        }
    };

    broadCastAddress = new Task(1*2000) {
        public void doTask() throws Exception {
            CommHandler.sendSpotAddress(Inputs*2);
        }
    };

    while(!done){showCount(Inputs);}
    CommHandler.openGeneralUseConnections();
    broadCastAddress.start();

    Ack = CommHandler.waitforAck();

    if (Ack){
        broadCastAddress.stop();
        CommHandler.closeGeneralUseConnections();
        CommHandler.openHostConnection();
        sysManager.createNNHandler(Inputs);
        CommHandler.closeHostConnection();
    }

    CommHandler.openP2PConnection("82");

    while (true){
        SPOTKnown = sysManager.poleSensorActivity();
        if (SPOTKnown){
            sysManager.gatherReadings();
        }
        if (!(processData.isActive())){
            processData.start();
            System.out.println("Starting data Processing");
        }
    }
```

Εικόνα 33: Ο κώδικας της κλάσης Aggregator.class.

Όπως φαίνεται στην Εικόνα 33, αρχικά το midlet με έναν while βρόχο, περιμένει από το χρήστη να καθορίσει το πλήθος των Sampler SPOT που καλείται το Aggregator SPOT να

χειριστεί, χρησιμοποιώντας το κουμπί SW1. Στην Εικόνα 34 φαίνεται η μέθοδος switchReleased η οποία λαμβάνει είσοδο από το χρήστη και ενημερώνει τα σχετικά μέλη της κλάσης και η μέθοδος showCount η οποία χειρίζεται τα led του SPOT και παρέχει οπτική ανάδραση στο χρήστη. Η επιλογή του μεγέθους εισόδου γίνεται με το πλήκτρο SW2.

```
public void switchReleased(SwitchEvent se) {

    if (!done){
        if (se.getSwitch() == sw1) {
            if (Inputs<4){
                Inputs+=1;
            }
            else if (Inputs>=4){Inputs = 1;}
            if (Inputs == 0){Inputs = 1;}
        }
        else {
            done = true;
            sysManager.neuralOutput = new double[Inputs];
        }
    }
    else{
        sysManager.getPool().printTanks();
    }
}

private void showCount(int count) {
    for (int i = 7, bit = 1; i >= 0; i--, bit <<= 1) {
        if ((count & bit) != 0) {
            leds.getLED(i).setOn();
        } else {
            leds.getLED(i).setOff();
        }
    }
}
```

Εικόνα 34. Οι μέθοδοι που ελέγχουν την είσοδο του χρήστη και την απεικόνιση αυτής στο Led array του SPOT.

Στη συνέχεια ξεκινάει μετάδοση της διεύθυνσης του SPOT - με το task broadcastAddress(); η οποία σταματάει μόνο όταν το SPOT λάβει μήνυμα Αναγνώρισης (ACK) από την host εφαρμογή. Αφού ληφθεί το ACK, κλείνει η broadcast σύνδεση και δημιουργείται μία νέα αφοσιωμένη σύνδεση με το basestation SPOT που είναι συνδεδεμένο στον υπολογιστή που εκτελεί την host εφαρμογή και αποστέλλεται αίτημα απόκτησης συναπτικών βαρών με τη μέθοδο (sysManager.createNNHanlder(Inputs)). Αφού αποκτηθούν τα συναπτικά βάρη, κλείνει η σύνδεση ανάμεσα στο Aggregator SPOT και την host εφαρμογή και δημιουργείται καινούργια σύνδεση στο port 82. Σε αυτή τη σύνδεση αποστέλλουν τα Sampler SPOT τις μετρήσεις τους. Σε

ένα αέναο βρόχο (while(){...}) ενεργοποιείται ένα καινούργιο task () το οποίο κάθε 1000 ms - τιμή που ορίζεται από το σχεδιαστή - ελέγχει εάν οι μετρήσεις που έχει λάβει μέχρι εκείνη τη χρονική στιγμή, πληρούν τα κριτήρια για να τροφοδοτηθούν στο νευρωνικό δίκτυο.

4.2.2. Coordinator.class

```
/**
 * Class responsible for gathering data from Sampler SPOT that operate under
 * Agrregator supervision. Instantiated by class NeuralSpotMidlet. Basically
 * this is the System Manager that instantiates all other classes and calls
 * all other methods required.
 * @author Michalis
 */
public class Coordinator{

    /**
     * Linked List containing SensroNodes. For each Sampler SPOT that connects
 with this Coordinator a SensorNode is created, holding Sampler's data.
     */
    private SensorSpotPool Pool = new SensorSpotPool();
    /**
     * Variable indicating whether the Sampler that just communicated is known
     * or not. If known then NeuralSpotMidlet class calls method to gather Data,
     * if not known then Sampler gets added in SensorNodePool as a SensorNode.
     */
    boolean currentSpotKnown = false;
    /**
     * The address of the Sampler SPOT that just communicated.
     */
    String currentSpotAddress;
    /**
     * Indicates how many of the Samplers to be entered have already entered
 Coordinator supervision.
     */
    int sensorsEntered = 0;

    /**
     * An object of class dataProcessor. dpr is responsible for processing
     * received data and calculating statistical values needed for neural
     * network.
     */
    dataProcessor dPr = new dataProcessor();

    /**
     * Value indicating if current data are above the statistical Thresholds
     * set in class dataProcessor.
     */
    boolean event = false;

    /**
     * An object of class neuralHandler. netHand is responsible for creating
     * neural network, loading received synaptic weights to the netwrok,
     * validating it and feeding current - when needed - to the neural network.
     */
    NeuralHandler netHand;
```

```
    /**
     * In case of neural network with input greater than one; i.e when more than
one Sampler's under Coordinator supervision, neural network output is this
array holding each of the output values for each of the inputs taken from
the input array.
     */
    double[] neuralOutput;

    /**
     * This is the neural network output after the Majority Voting.
     */
    double finalNeuralOutput = 0;

    /**
     * In case of neural network with input greater than one; i.e when more than
one Sampler's under Coordinator supervision, neural network input is this
array holding data from all Samplers in pairs.
     * input = [t1,h1,t2,h2,t3,h3]
     */
    private Vector input = new Vector();

    /**
     * Threshold value for majority voting. If neural output above this value
     * then neural output is equal to 1.
     */
    private final double normalizationThreshold = 0.5;


    /**
     * Waits to receive data from some spot. Once data received
     * either adds the tank the pool or informs that tank is already
     * known.
     * @return
     */
    public boolean poleSensorActivity(){}

    /**
     * Calls SensorSpotPool method responsible for storing received data.
     */
    public void gatherReadings(){}

    /**
     * @return the pool this midlet has. Pool is consisted
     * by tanks. A tank is the representation of a SensorSpot
     * in this midlet. Each tank among others has two dataqueues
     * containing temperature and humidity readings form this
     * specific sensorSpot.
     */
    public SensorSpotPool getPool(){}

    /**
     *
     * @return the number of sensors (Sampler SPOT) that entered
     * NeuralSPOT network.
     */
    public int getsensorsEntered(){}

    /**
     * Calls class dataProcessor methods that handle data statistical analysis
     * and scenario checking. First Pool iteration is synchronised so no storing
     * methods can access pool. Second pool iteration checks the result of
     * iteration 1, to see if any of Samplers indicates fire conditions.
     */
```

```
    public void checkForFire(){}

    /**
     * Instantiates an object of class NeuralHandler. Basically this is where the
     * neural network is being created.
     * @param Inputs. The number of inputs chosen by User multiplied by two.
     * Input size of 1 equals to one Sampler. One Sampler equals to two measure-
     * ments. One for temperature and one for humidity.
     */
    public void createNNHandler(int Inputs){}

    /**
     * Finds last data received [t&h] from each Sampler and stores them in a
     * Vector.
     * @param pool SensorSPOT list that contains tanks. A tank is a sensor
     * representation. Each Tank holds two dataqueues. One for temp one for
     * humidity.
     * @return A Vector containing the last temperatures received and
     * corresponding humidities.
     */
    public Vector constructNNInputVector(SensorSpotPool pool){}

    /**
     * Normalizes input var if input val above normalizationThreshold.
     * @param input. It is the neural network output array holding values anywhere
     * between 0 and 1.
     */
    public void normalizeNeuralOutput(double[] input){}

    /**
     * Majority voting on neural network output.
     * @param input.Neural network output after normalization. This array holds
     * values that are either 1 or 0.
     * @return a single double value indicating the value held by the majority
     * of the input array elements.
     */
    public double MajorityVoteOutput(double[] input){}

    /**
     * Handles the neural network raw output and resets neural network input.
     * It normalizes the networks raw output to either 0 or 1 and then passes
     * the normalized values to majority voting algorithm. Prints the results.
     * @param output. Neural networks raw output.
     */
    public void handleNNOutput(double[] output){}
}
```

4.2.3. NeuralHandler.class

```
/**
 * Class responsible for creating training feeding the neural network deployed
 * on the NeuralSpot. This class is instantiated and accessed only by Aggregator
 * class.
 * @author Michalis
 */
public final class NeuralHandler {

    /**
     * Variable holding normalized humidity data for NN input.
     */
    NormalizedField HumInput = new NormalizedField(NormalizationAction.Normalize,
                                                    "Hum",99.8,10.004, 1,0);

    /**
     * Variable holding normalized temperature data for NN input.
     */
    NormalizedField TempInput = new NormalizedField(NormalizationAction.Normalize,
                                                    "Temp",50,5.2, 1,0);

    /**
     * The networks standard input layer neuron count.
     */
    private final int stdNetworkInputSize = 2;

    /**
     * Input Stream for validation file which will be used after synaptic
     * weights received.
     */
    private InputStream InpS = getClass().getResourceAsStream("/validationDataset_"+
                                            (stdNetworkInputSize/2)+"dNet.txt");;
    /**
     * Midlets neural network.
     */
    private final BasicNetwork network;

    /**
     * counter measuring the number of weights received so far.
     */
    private static int weightsIndex = 0;

    /**
     * The number of elements the neural network weight matrix holds.
     * Basically this is the count of synaptic weights that this Aggregator
     * is expecting to receive from host app.
     */
    static int weightArraySize = 0;

    /**
     * The actual synaptic weights.
     */
    static Double[] modelWeights;
```

```
    /**
     * The user defined input size for this network. Regardless of this number
     * the network will be created according to standard architecture (2 input
     * neurons. BUT the input for these two neurons will be constructed according
     * to this number. Example: User input equal to 3 informs that the network
     * should have 6 input neurons. Instead network will have 2 neurons and input
     * will be an array of size networkInputSize*2 holding the 3 pairs of data.
     * Input will be fed serially to the network, one pair at a time.
     */
    private int networkInputSize = 0;

    /**
     * Class NeuralHandler Constructor
     * @param networkInputSize: is the activeSize of the network to be created.
     */
    public NeuralHandler (int networkInputSize){}

    /**
     * Creates the network. Has to be the same as the one in host application
     * @return the network
     */
    public BasicNetwork createNetwork(){}

   /**
    * Feed network with the current set of data.
    * @param Input. A vector holding all input data pairs [h+t]
    * @param activeSize. The number of active Samplers. If input size equals
    * to 3 yet only one of the samplers is currently active then active size
    * equals to 1. So input array will have only one pair of valid data and
    * two pairs of dummy data.
    * @return
    */
    public double[] feedNetwork(Vector Input, int activeSize){}

    /**
     * Tests the network as it is configured with the obtained weights.
     */
    public  void testNetwork(){}

    /**
     * @return The activeSize of NeuralNets Weight Matrix
     */
    public int getModelWeightsMatrixSize(){}
}
```

4.2.4. dataProcessor.class

```
/**
 * A class responsible for analyzing data acquired by the SPOTensors.Instantiated
 * and accessed only by Aggregator class.
 * @author Michalis
 */
public final class dataProcessor {

   /**
    * For each Sampler, we compute the weighted moving average for the values
    * held at each of the queues [one queue for temp, one for humidity] that
    * Sampler implements. This is the sum of the weights applied to the values of
    * the queues.
    */
    private int weightedAverageWeightSum = 15;

    /**
     * The sum of the five values of temperature held by a SensorNode in
     * temperature dataQueue.
     */
    double sumTemp;

    /**
     * The sum of the five values of humidity held by a SensorNode in
     * temperature dataQueue.
     */
    double sumHum;

    /**
     * A sensorNodes Max Temperature for the current set of data.
     */
    double maxTemperature;

    /**
     * A sensorNodes corresponding Humidity for the Max Temperature.
     */
    double maxTHumidity;

    /**
     * A sensorNodes Min Humidity for the current set of data.
     */
    double minHumidity;

    /**
     * A sensorNodes corresponding Temperature for the Min Humidity.
     */
    double minHTemperature;

    /**
     * The threshold above which a temperature reading is considered to indicate
     * fire conditions. Clearly experimental.
     */
    double AvTempThreshold;

    /**
     * Threshold value above which data is considered suspicious and fed to NN.
     * Direction is measured with an integer value indicating how many sets of
     * data have been processed so far that are consisted of increasing values.
     */
    int tempDirectionThreshold;
```

```
/**
 * Threshold value above which data is considered suspicious and fed to NN.
 * Steep fluctuations in temperature are considered suspicious. A deviation
 * of 5 would suggest a rise in temperature of more than 4°C between
 * readings.
 */
double temperatureDeviationThreshold;

/**
 * Threshold value above which data is considered suspicious and fed to NN.
 * Steep fluctuations in humidity are considered suspicious. This value is
 * set experimentally to 15 due to lack of real data.
 */
double humidityDeviationThreshold;

/**
 * Value indicating the size of the Dataque holding temperature/humidity
 * readings. Used for average, direction and deviation calculation
 */
private int tankDatapoolSize;

/**
 * State refers to FSM.
 */
static int state ;

/**
 * This is one of FSM States. Idle state signifies that all data are within
 * desired range.
 */
static public final int FSM_STATE_IDLE=0;

/**
 * This is one of FSM States. Deviation_Fire signifies that Deviation for
 * current set of temperature data is above the corresponding threshold and
 * the temperature is increasing.
 */
static public final int FSM_STATE_DEVIATION_FIRE=2;

/**
 * This is one of FSM States. WMAVERAGE_FIRE signifies that the weighted moving
 * average is below the current last temperature reading which is above 28°C
 */
static public final int FSM_STATE_WMAVERAGE_FIRE=3;

/**
 * Neural Network is fed for 10 runs regardless of the neural network output.
 * After the 10th network run, fsm state resets to idle. Then if the data are
 * still above threshold fsm state changes to the corresponding state and the
 * network fires again for another set of 10 runs. If data are below thresholds
 * fsm state remains Idle.
 */
static int maxConseqReadings = 9;

/**
 * Counter that holds the value of runs made by network on a set of data.Once
 * this variable reaches 9, FSM returns to Idle state and will reiterate if
 * needed.
 */
static int conseqReadsIndex = 0;


/**
```

```
     * Class constructor. Initializes instance variables.
     */
    public dataProcessor(){}

    /**
     * Calls all methods responsible for calculating statistical figures on data
     * obtained so far.
     * @param pool is a linked list of tanks where tank is the representation
     * of a sensorSpot in this midlet. Each tank contains a queue for temp
     * and a queue for humidity.
     */
    public void processData(SensorNode tank){}

    /**
     * In order to achieve less dataqueue iterations minMax values
     * and temperature/humidity sums are being calculated here. Results
     * are stored in class members and in tank with setters.
     * @param tank is the current spot from sensorSpotPool
     */
    private void getStatistics(SensorNode tank){}

    /**
     * Check in which direction the average temperature
     * is moving towards. Can be either rising or falling
     * @param tank is the current spot from sensorSpotPool
     */
    private void setAverageDirection(SensorNode tank){}

    /**
    * Returns the running average for each tank both in humidity
    * and temperature. It iterates the whole list each time
    * but could be made faster.
    * @param tank ::is the current spot from sensorSpotPool
    */
    private void setWMAverage(SensorNode tank){}

    /**
     * Calculates the tanks data standard deviation.
     * @param tank is the current spot from sensorSpotPool.
     */
    private void getStdDeviation(SensorNode tank){}

    /**
     * Sets the tanks max temperature and the corresponding humidity
     * Sets the tanks min humidity and the corresponding temperature
     * @param tank is the current spot from sensorSpotPool.
     */
    private void setMinMaxValues(SensorNode tank){}

    /**
     * Class is static so variables need to be reseted to zero
     * for next runs not to include previous calculations.
     */
    private void resetVars(){}

    /**
     * Passes all tanks to poleforEvent();
     * @param pool
     * @return True if one of the tanks data satisfies fire conditions
     */
    public boolean scanForEvent(SensorNode tank){}
```

```
    /**
     * Checks to see in what state the midlet is.
     * Idle:: There are no measurements indicating fire.
     * FSM_STATE_DEVIATION_FIRE:: abnormal std deviation behaviour that could
     * indicate fire.
     * FSM_STATE_WMAVERAGE_FIRE:: last t reading above threshold and above
     * WMaverage. Possible fire.
     * @param tank
     * @return true if NOT in Idle state, false if otherwise.
    */
    private boolean poleforEvent(SensorNode tank){}

    /**
     * Checks the conditions assigned to fire indication.
     * @param tank is the current Spot from SpotPool (pool passed in processData)
     * @param input a string indicating whether check should be for temperature
     * or abnormal standard deviation behavior.
     * @return true if conditions check, false otherwise.
     */
    private boolean checkFireConditions(SensorNode tank,String input){}
}
```

4.2.5. CommHandler.class

```
/**
 * Class responsible for communication handling between HostApp-NeuralSpot and
 * NeuralSpot- freeRangeSPOT. Static class needs not to be instantiated.
 */
public class CommHandler {

    /**
     * Opens connection with host App. One connection broadcasts. Other one
     * receives form hostApp.
     */
    public static void openGeneralUseConnections(){}

    /**
     * Opens connection that will be used to send address to host App. Used
     * because at first run basestationSPOT used by hostApp address is not
     * know. Port used is 83.
     */
    public static void openBroadcastConnection(){}

    /**
     * Connection used to receive from host. Port used 83.
     */
    public static void openP2PConnection(){}

    /**
     * Connection used to communicate with FreeRangeSPOT. Port used is 82.
     * @param Port the port to be used for NeuralSpot<->FreeRangeSpot comms.
     */
    public static void openP2PConnection(String Port){}

    /**
     * If HostAddress know this connection can be used in order to avoid using
     * BroadcastConnection() and P2P()
     */
    public static void openHostConnection(){}

    /**
     * Closes P2P() and BroadcastConn()
     */
    public static void closeGeneralUseConnections(){}

    /**
     * Halt here until an Ack message is received.
     * @return Boolean indicating Ack value
     */
    public static boolean waitforAck(){}

    /**
     * Once powered on the NeuralSpot will start sending its own address until
     * the hostApp acknowledges it.
     * @param networkInputSize is the size of the network the NeuralSpot is
     * programmed to handle. Can be 2/4/6/8
     */
    public static void sendSpotAddress(int networkInputSize){}

     /**
     * Gets the size of the weights Matrix for a given network configuration
     * @return Int indicating the size of the matrix to be created.
     */
    public static int getMWMSize(){}
    /**
```

```
     * Gets the weights of neuralNetworks matrix one double at a time.
     * This method does not receive the whole matrix at once but one element
     * at a time. So it must be called matrixRows*matrixColumns times in order
     * to receive all the weights.
     * @return a double representing a weight in the neural network matrix.
     */
    public static double getMWeights(){}

    /**
     * Close Connection with HostApp. This happens once all matrix weights
     * are received.
     */
    public static void closeHostConnection(){}

    /**
     * Halt here until a data message from a freeRangeSpot is received.
     */
    public static final void waitforFreeRangeSpot(){}

    /**
     * Sends an Ack message to HostApp
     */
    public static final void sendAck(){}

    /**
     * freeRangeSpot Status can be either true(known spot) or unknown(false).
     * If FRSpot is unknown than neuralSpot adds the FRSpot to the pool and then
     * sends an ack to the FRSpot. Next FRSpot message will have a "known"
     * status.
     * @return Boolean indicating the FRSpot status
     */
    public static boolean getFreeRangeSpotStatus(){}

    /**
     * Receives the FRSpot address.
     * @return a String representing the last 8 digits of the FRSpot address.
     */
    public static String getFreeRangeSpotAddress(){}

    /**
     * Gets a reading from an FRSpot.
     * In the case of each FRSpot holding two readings, this method must be
     * called two consecutive times. First time receives humidity reading
     * second time receives temperature reading.
     * @return A double holding the current reading received.
     */
    public static double getReading(){}
}
```

4.2.6. SensorSpotPool.class

```
/**
 * Class that implements a linked list to hold the information for each FRSpot.
 * @author Michalis
 */
public  final class SensorSpotPool extends LinkedList{}

    /**
     * Adds a new tank (where tank is an FRSPot and FRSpot is represented by a
     * SensorNode).If list is empty it adds the tank first if not, it adds the
     * tank last
     * @param Id :: A string containing the FRSPOT'sSPOT address last 8 digits. It
     * is used as an identification between tanks.
     * @return a boolean indicating if the new tank with the specified Id is
     * already in the list
     */
    public boolean addTank(String Id){}

    /**
     * Gets the readings from the spot that currently
     * communicates with this NeuralSpot Midlet and then
     * stores them to tank's dataqueue.
     */
    public void pushCurrentData(String tankAddress){}

    /**
     * When a FRSPot communicates with neuralSpot and sends data, neuralSpot
     * must find the corresponding tank in order to store the data.
     * Identification is implemented with the use of Id
     * @param Id is the currently communicating FRSpot address
     * @return A reference to the tank that corresponds to the currently
     * active FRSpot
     */
    public SensorNode getCurrentTank(String Id){}

    /**
     * Checks if a tank with Id is already on the list. Basically, an already
     * known FRSpot could stop transmiting (reset/power failure) and then
     * reenter the network. In case of reentrance there is no need for a new
     * tank since there already is one for the specific FRSpot that
     * also carries the FRPsots data history
     * @param Id The identification token.
     * @return A boolean that is True if the Id is not known to the
     * list, false otherwise
     */
    private boolean checkTank(String Id){}

    /**
     * Prints all of the tanks currently in the list among with the data that
     * each tank holds.
     */
    public  void printTanks(){}
}
```

4.2.7. Dataqueue.class

```
/**
 * Class DataQueue implements the queues that hold the humidity
 * and temperature data for each SensorNode (FRSpot). It is implemented
 * using a linked list which once has reached the desired number of data
 * entries(by adding nodes), just pushes new data and pops old. In this case
 * push means remove data from head. Enter new data. Move head to tail. Make
 * head->next the new head.
 * @author Michalis
 */
public class DataQueue extends LinkedList{

    /**
     * The last element in this Queue.
     */
    private DataNode tail;

    /**
     * Queue has a max size of 5. This variable counts the number of elements
     * that have been added to the queue. Once current size is equal to maxSize
     * queue allows only push pop operations.
     */

    private int currentSize;

    /**
     * The maximum size this queue is allowed to have.
     */
    private int maxSize;

    /**
     * A flag indicating whether the queue is full or not.
     */
    private boolean queueFullFlag = false;
   /**
     * Class constructor.
     */
    public DataQueue() {}

    /**
     * Indicates whether this queue is empty or not.
     * @return True if queue is empty, false otherwise.
     */
    public boolean isEmpty(){}

    /**
     * Returns the queue current size.
     * @return integer indicating the queue current size.
     */
    public int getCurrentSize(){}

    /**
     * Getter Method.
     * @return the max size allowed for the queue to take
     */
    public int getMaxSize(){}

    /**
     * Setter Method. Sets the maximum size the queue can take.
     * @param maxsize Integer value indicating the maximum size.
     */
    public void setMaxSize(int maxsize){}
```

```
    /**
     * Pushes an element at queue last position. Here queue is implemented as a
     * linked list. If list consists from less than maxSize nodes then just add
     * the new node in the last position. If list consists of maxSize nodes, then
     * put new data in head position and make head the new tail.
     * @param data The new data that needs to be pushed into the queue
     */
    public void push (double data){}

    /**
     * Removes queue's first element.
     */
    private void pop(){}

    /**
     * Setter Method.
     * Sets a flag indicating that the queue is full
     */
    public void setQueueFullFlag(){}

    /**
     * Returns a value indicating whether the queue is full or not.
     * @return boolean value.
     */
    public boolean getQueueFullFlag(){}

    /**
     * Returns the data held by the queue's last element.
     * @return a double holding the tails data.
     */
    public double getTailData(){}

    /**
     * Returns a reference to Datqueues last element.
     * @return a Datanode var holding a reference to tail element.
     */
    public DataNode getTail(){}
}
```

4.2.8. LinkedList.class

```
/**
 * Class implementing a linled list due to the fact that j2me
 * does not suppoert such a structure.
 * @author Michalis
 */
public class LinkedList {

    /**
     * A variable holding a reference to the first element in the list. Used
     * for list iteration.
     */
    private Node head;

    /**
     *  Creates a new node to be inserted in the linked list.
     * @param input. A string if node to be created is of SensorNode class, a
     * double if node is of class DataNode.
     * @return a reference to the node created.
     */
    private Node createNode(Object input){}

    /**
     * LinkedList Method. Places a new node in linkedlist @ head position
     * @param Input Can be either DataNode or SensorNode. Downcasting will be
     * needed.
     */
    protected void insertFirst(Object Input){}

    /**
     * LinkedList Method. Places a new node in linkedlist @ last position
     * @param Input Can be either DataNode or SensorNode. Downcasting will be
     * needed.
     */
    protected void insertLast(Object Input){}

    /**
     * Removes the node at the first place.
     * @return A node taking the head place.
     */
    protected Node removeFirst(){}

    /**
     * Remove the last node in the list.
     * @return the node that was just removed if list not empty, return the
     * head otherwise
     */
    protected Node removeLast(){}

    /**
     * Get list's head Node.
     * @return head Node (can be either SensorNode or DataNode)
     */
    public Node getHead(){}

    /**
     * Set the given Node as the head of the list
     * @param newHead the Node to be placed at head position
     */
    protected void setHead(Node newHead){}
}
```

4.2.9. SensorNode.class

```
/**
 * SensorNode class is used to create a linked list called SensorSpotPool.
 * NeuralSpot can have 1,2,3 or 4 FRSpots under its supervision. The FRSpots
 * under the neuralSpot supervision are represented as SensorNodes. Each FRSpot
 * is a sensorNode. SensorNodes are used to create a linked list of type
 * SensorSpotPool. At each SensorNode there are members that hold key data
 * for analysis to match fire criteria. For example AverageTemp is the
 * SensorNodes (remember a sensorNode IS a FRSpot) average temperature.
 * average temperature is measured by adding all the data stored in the
 * temperature queue and dividing by the number of entries.
 * @author Michalis
 */
public final class SensorNode extends Node{

    /**
     * A Dataqueue holding this SensorNodes last five
     * temperature readings.
     */
    public DataQueue temperatureData;

    /**
     * A Dataqueue holding this SensorNodes last five
     * humidity readings.
     */
    public DataQueue humidityData;

    /**
     * Class member indicating how many tanks(SensroNode) belong to this pool.
     */
    public static int numofTanks = 0;

    /**
     * Number indicating the order in which this SensorNode entered.
     */
    private final int tankNumber;

    /**
     * String that holds this Tanks address last 6 digits. Used to find
     * specific tank among others.
     */
    public final String idAddress ;

    /**
     * Tanks [Weighted Moving]Average Humidity.
     */
    private double averageHum;

    /**
     * Tanks [Weighted Moving]Average Temperature.
     */
    private double averageTemp;

    /**
     * Tanks humidity standard deviation.
     */
    private double standDevHum ;

    /**
     * Tanks temperature standard deviation.
     */
```

```
    private double standDevTemp;

    /**
     * Tanks max temperature.
     */
    private double maxTemperature;

    /**
     * Tanks corresponding humidity for MAX temperature.
     */
    private double maxTHumidity;

    /**
     * Tanks minimum humidity.
     */
    private double minHumidity;

    /**
     * Tanks corresponding temperature for minimum humidity.
     */
    private double minHTemperature;

    /**
     * The direction the tank's s temperature weighted moving average follows.
     */
    private int averageTempDirection;

    /**
     * Boolean indicating whether fire occurrence or not. True = fire
     * False = no fire.
     */
    private boolean event;

    /**
     * Class Constructor. Initializes variables.
     * @param Id the address of the Sampler this SensorNode represents.
     */
    public SensorNode(String Id) {}

    /**
     * Get the tank event state. If tank matches fire criteria, event - true
     * otherwise event = false.
     * @return boolean.
     */
    public boolean getTankEvent(){}

    /**
     * Set tank event state at the desired state.
     * @param state boolean holding the desired state.
     */
    public void setTankEvent(boolean state){}

    /**
     * Sets the direction in which temperature is moving.
     * If averageTempDirection is positive, temperature is increasing.
     * Temperature is decreasing otherwise.
     * @param data integer indicating the direction. If data is nonZero
     * data is added to the current direction. If data is zero than direction
     * is zero which means that temperature just changed direction.
     */
    public void setAverageTempDirection(int data){}
```

```
    /**
     * Return the FRSpot temperature direction.
     * @return An integer indicating the temperature direction
     * positive number: temp is increasing.
     * negative number: temperature is decreasing.
     * zero: temperature just changed direction.
     */
    public int getAverageTempDirection(){}

    /**
     * Returns the FRSpot max temperature.
     * @return A double representing the temperature.
     */
    public double getMaxTemperature(){}

    /**
     * Sets the FRSpot Max Temperature to a specified value
     * @param data double that holds the value of the desired temperature.
     */
    public void setMaxTemperature(double data){}

    /**
     * Returns the FRSpots humidity reading that corresponds to FRSpots max
     * temperature.
     * @return double holding the humidity value.
     */
    public double getMaxTHumidity(){}

    /**
     * Set the humidity value that corresponds to this FRSpot;s max
     * temperature.
     * @param data A double hilding the desired humidity value.
     */
    public void setMaxTHumidity(double data){}

    /**
     * Returns FRSpots minimum humidity.
     * @return double indicating the minimum humidity value
     */
    public double getMinHumidity(){}

    /**
     * Set the the FRSpot's minimum humidity to a desired value
     * @param data a double hilding the desired value
     */
    public void setMinHumidity(double data){}

    /**
     * Return the temperature that corresponds to the FRSpot's minimum humidity
     * @return A double holding the temperature value
     */
    public double getMinHTemperature(){}

    /**
     * Set the temperature that corresponds to FRSpot's minimum humidity
     * to a desired value.
     * @param data A double holding the desired value
     */
    public void setMinHTemperature(double data){}

    /**
     * Returns the FRSpot's Temperature standard Deviation.
     * @return A double holding the deviation
```

```
     */
    public double getstandDevTemp(){}

    /**
     * Returns the FRSpot's Humidity standard Deviation.
     * @return A double holding the deviation
     */
    public double getstandDevHum(){}

    /**
     * Set the FRSpot's Humidity standard Deviation to a desired value.
     * @param data A double holding the desired value
     */
    public void setstandDevHum(double data){}

    /**
     * Set the FRSpot's Temperature standard Deviation to a desired value.
     * @param data A double holding the desired value
     */
    public void setstandDevTemp(double data){}

    /**
     * A number indicating the order in which the FRSpot entered the network
     * @return An integer holding the order value
     */
    public int getpoolNumber(){}

    /**
     * Get the FRSpot average humidity
     * @return A double holding the value
     */
    public double getWMAverageHum(){}

    /**
     * Get the FRSpot average temperature
     * @return A double holding the value
     */
    public double getWMAverageTemp(){}

    /**
     * Set FRSpot's average humidity to a desired value
     * @param data A double holding the desired value
     */
    public void setWMAverageHum(double data){}

    /**
     * Set FRSpot's average temperature to a desired value
     * @param data A double holding the desired value
     */
    public void setWMAverageTemp(double data){}
}
```

4.2.10. DataNode.class

```
/**
 * DataNode class is used for the creation of queues. Each FRSpot that belongs
 * to NeuralSpots supervision is represented in the NeuralSpot as a
 * SensorNode(Class). Each SensorNode holds data for humidity and temp. This
 * data is hold in a queue for humidity and a queue for temp. A queue is made
 * using a linked list. And the linked list used for the queues is using
 * Datanodes as a dataStructure.
 * queue.
 * @author Michalis
 */

public class DataNode extends Node{

    /**
     * The temperature or humidity data that this node will hold.
     */
    private double data;

    /**
     * Class Constructor
     * @param data. The temperature or humidity data that this node will hold
     */
    public DataNode(double data) {}

    /**
     * Set a defined value (double value representing humidity or temperature)
     * to the DataNodes data member.
     * @param data The data to be stored
     */
    public void setData(double data){}

    /**
     * Getter method.
     * @return The data this node holds.
     */
    public double getData(){}
}
```

4.2.11. Node.class

```
/**
 * Base Class is used to implement SensorNode and DataNode
 * @author Michalis
 */
public class Node {
    /**
     * a pointer to the next node in the list
     */
    private Node next;

    /**
     * Class Constructor. Base element for linked list class.
     */
    public Node(){}

    /**
     * Sets the "next pointer" to the succeeding node
     * @param next. A reference to the succeeding node
     */
    public void setNext(Node next){}
```

```
    /**
     * Returns a reference to this node's succeeding node.
     * @return
     */
    public Node getNext(){}
}
```

4.2.12. StringUtils.class

```
/**
 * Class that contains complementary methods
 * @author Michalis
 */
public final class StringUtils {

    /**
     * Splits a given string to a set of substrings according to the delimiter
     * given
     * @param original. The initial String that will be splited
     * @param separator. The delimiter on which the String will be split.
     * @return an array of Strings containing all the "sub" Strings
     */
    public static String[] split(String original,String separator) {}
}
```

4.3. Sampler Midlet

Είναι το midlet που εκτελείται στα Sampler SPOT. Εκτελεί δυο βασικές λειτουργίες. Η πρώτη είναι η μετάδοση της διεύθυνσης του SPOT στο οποίο εκτελείται και η δεύτερη είναι η μετάδοση μετρήσεων προς το Aggregator SPOT στου οποίου την εποπτεία ανήκει. Οι μετρήσεις που αποστέλλει το Sampler στο Aggregator, δεν είναι πραγματικές. Υπάρχουν στο φάκελο \resources τρία αρχεία τύπου .txt, καθένα από τα οποία περιέχει διαφορετικές μετρήσεις. Τα βήματα που ακολουθούντα είναι τα εξής:

1. Αναμονή για είσοδο από το χρήστη. Ο χρήστης, χρησιμοποιώντας το κουμπί SW1 επιλεγεί ποιο από τα 3 διαθέσιμα αρχεία μετρήσεων θα χρησιμοποιήσει το Sampler. Η επιβεβαιώσει επιλογής γίνεται με το κουμπί SW2.
2. Αφού επιβεβαιωθεί η επιλογή του χρήστη, το Sampler εκπέμπει τη διεύθυνση του και στη συνέχεια αναμένει παραλαβή μηνύματος αναγνώρισης (ACK) από το Aggregator SPOT.
3. Αφού ληφθεί το μήνυμα αναγνώρισης, φορτώνεται στη μνήμη ολόκληρο το αρχείο μετρήσεων και εκκινείτε το task που είναι υπεύθυνο για την αναμετάδοση των μετρήσεων
4. Κάθε 500 ms (παραμετροποιήσιμος χρόνος) το task διαβάζει μία σειρά από το αρχείο που έχει στη μνήμη του και αποστέλλει το ζεύγος μετρήσεων στο Aggregator Spot.
5. Όταν αποσταλούν όλες οι μετρήσεις από το αρχείο το task τερματίζεται και το Sampler παραμένει σε αδράνεια.

4.3.1. Sampler.class

```
/**
 * Class that implements Sampler SPOT.
 * 1.- Broadcast its address in order to become known
 * 2.- Send data in time interval defined by administrator.
 */
public class Sampler extends MIDlet implements ISwitchListener{

    Task readSensor = null;

    /**
     * Boolean value indicating whether this Sampler is known to AggregatorSPOT.
     */
    boolean knownSensor = false;

    /**
     * Boolean value indicating whether input file is chosen or not.
     */
    boolean done = false;

    /**
     * USER adjusted integer representing the file number which will be chosen.
     * Handled by switchReleased() method.
     */
    private int Inputs = 0;

    /**
     * The current line from the input file.
     */
    String currentLine;

    /**
     * When this index equals input file lines number, broadcasting data stops.
     */
    int dataFileIndex = 0;

    /**
     * Stringbuffer that stores all lines from input file.Appends a \n after
     * each line
     */
    StringBuffer stringBuffer = new StringBuffer();

    /**
     * The Aggregator SPOT address.
     */
    private final String AggregatorAddress = "C0A8.8480.0000.1001";

    /**
     * The file chosen by the user via SW1 SW2.
     */
    private String inputFile;

    /**
     * Port used for communication between Sampler and Aggregator.
     */
    private static final int HOST_PORT = 82;

    protected void startApp() throws MIDletStateChangeException {

        /**
         * A task that reads a line from stringBuffer, splits line on delimiter
         * and then uses connection to send data to Aggregator.
```

```
         */
        readSensor = new Task(1*500) {};
        /**
         * while SW2 has never been pressed handle led's to represent
         * users input.
         */
        while(!done){showCount(Inputs);}

        /**
         * As long as this Sampler is not know to the Aggregator, Sampler
         * must broadcast address until ACK from Aggregator received.
         */
        while (!knownSensor) {
                /**
                 * Once acknowledged get data from input file
                 * and start the task responsible for reading and transmitting
                 * the data.
                 */
                if (knownSensor) {
                    getInputFile();
                    readSensor.start();
                }
        }
    }

    /**
     * Sets a led on once the Sampler is known to the Aggregator.
     */
    private void setNetworkLed(){}

    /**
     * Led indicating that the Sampler is operational.
     * @param Led. The Led number to toggle.
     */
    private void toggleLedIndic(int Led){}

    /**
     * Reads all data from input file and stores it to a stringBuffer
     * @param fileReader the file containing the data.
     * @return the data to an array of strings.S
     */
    public String[] getDatafromFile(BufferedReader fileReader) {}

    /**
     * If SW1 released then handle input file choise - if SW2 then "enter"
     * @param se
     */
    public void switchReleased(SwitchEvent se) {}

    /**
     * User input.
     */
    private void getInputFile(){}
    /**
     * Handles the SPOT Led Array to show user Input.
     * Leds toggle to repsresent count value (1 - 4) in binary form.
     * @param count. The number of Sampler SPOT this Coordinator will supervise.
     * Represented on the SPOT led array as a binary value.
     */
    private void showCount(int count) {}
}
```

5. Συμπεράσματα - Βελτιώσεις.

5.1. Βελτιώσεις

Το σύστημα που υλοποιήθηκε, βασίζεται σε νευρωνικό δίκτυο για την ανίχνευση πυρκαγιάς, βασίζεται ακόμα έστω και περιορισμένα και σε τιμές κατωφλίου. Είδαμε στην παράγραφο 2 του κεφαλαίου 3.3 (Επίπεδο Συλλογής και Επεξεργασίας) ότι το νευρωνικό δίκτυο θα τροφοδοτηθεί με μετρήσεις μόνο εάν διαπιστωθεί υπέρβαση των τιμών κατωφλίου είτε της τυπικής απόκλισης είτε της θερμοκρασίας. Το μειονέκτημα εδώ είναι ότι τέτοιες τιμές κατωφλίου δεν μπορούν να είναι αντιπροσωπευτικές για όλο το έτος δεδομένου ότι η θερμοκρασία και ο ρυθμός μεταβολής αυτής μέσα στην μέρα αλλάζουν ανάλογα με την εποχή. Για παράδειγμα μια τιμή κατωφλίου στους 28°C για την θερμοκρασία, μπορεί το φθινόπωρο και τον χειμώνα να καλύπτει τις ανάγκες του συστήματος, την άνοιξη όμως και το καλοκαίρι, ενδεχομένως να προκαλεί πολλούς εσφαλμένους συναγερμούς δεδομένου ότι 28°C είναι μια φυσιολογική θερμοκρασία καλοκαιρινού μήνα. Παράλληλα ακόμα και αν τα δεδομένα εκπαίδευσης περιλαμβάνουν μετρήσεις που καλύπτουν όλη την διάρκεια του έτους, οι κλιματολογικές συνθήκες σε μια περιοχή αλλάζουν με την πάροδο του χρόνου. Θα ήταν χρήσιμη επομένως η ανάπτυξη ενός συστήματος που μπορεί να αναγνωρίσει αυτές τις αλλαγές και να προσαρμόσει τόσο τις τιμές κατωφλίου στις οποίες βασίζεται όσο και το νευρωνικό του δίκτυο, στα δεδομένα που λαμβάνει κατά την λειτουργία του. Παρακάτω παρουσιάζεται μια υλοποίηση που προσπαθεί να απαντήσει τα θέματα που αναφέρθηκαν η οποία ουσιαστικά αποτελεί επέκταση της λύσης που προτείνουν οι Yu και Wang [21].

Αρχιτεκτονική Συστήματος

Η αρχιτεκτονική του προτεινόμενου συστήματος δεν αλλάζει από την αρχιτεκτονική που τηρείται στην παρούσα υλοποίηση. Πλήθος κόμβων αισθητήρων κατανέμεται στην περιοχή που παρακολουθείται. Οι κόμβοι αυτοί χωρίζονται σε clusters και κάθε cluster έχει ένα κόμβο επικεφαλή. Οι κόμβοι αισθητήρες λαμβάνουν μετρήσεις υγρασίας και θερμοκρασίας περιβάλλοντος και τις αποστέλλουν στον Επικεφαλή κόμβο (ΕΚ). Ο ΕΚ αποφασίζει για την ύπαρξη ή μη ύπαρξη πυρκαγιάς μέσω ενός νευρωνικού δικτύου, του οποίου τα βάρη λαμβάνει από τον κόμβο Διαχειριστή (ηλεκτρονικός υπολογιστής) με τον οποίο συνδέεται ασύρματα.

Κόμβοι Αισθητήρες (Sampler)

Οι Αισθητήρες κόμβοι (ΑΚ) διατηρούν την ίδια λειτουργία με αυτή που έχουν στην παρούσα εργασία. Λαμβάνουν μετρήσεις υγρασίας και θερμοκρασίας περιοδικά και αποστέλλουν τις μετρήσεις αυτές στον Επικεφαλή κόμβο του cluster στο οποίο ανήκουν.

Επικεφαλής Κόμβοι (Aggregator)

Κάθε ΕΚ, αποθηκεύει τα δεδομένα που λαμβάνει και αν κριθεί από τον ίδιο απαραίτητο (υπέρβαση κάποιας τιμής κατωφλίου είτε στην θερμοκρασία είτε στην τυπική απόκλιση αυτής) τροφοδοτεί τα δεδομένα που έλαβε στο νευρωνικό δίκτυο που υλοποιεί προκειμένου να παρθεί η απόφαση για την ύπαρξη η μη ύπαρξη πυρκαγιάς. Το νευρωνικό δίκτυο που υλοποιείται εδώ δεν εκπαιδεύτηκε τοπικά αλλά στον Διαχειριστή κόμβο (ΔΚ) του συστήματος. Τα συναπτικά βάρη του νευρωνικού δικτύου τα παρέλαβε ο ΕΚ μετά από αποστολή αιτήματος συναπτικών βαρών στον ΔΚ. Παράλληλα, ο ΕΚ για κάθε ΑΚ του cluster το οποίο επιβλέπει, υπολογίζει και αποθηκεύει τις μέγιστες και ελάχιστες τιμές για υγρασία, θερμοκρασία και την τυπική απόκλιση θερμοκρασίας. Στην συνέχεια αποστέλλει στον ΔΚ το σύνολο των μετρήσεων και τις ελάχιστες/μεγιστες τιμές, που έχει στην μνήμη του για κάθε ΑΚ που επιβλέπει.

Διαχειριστής Κόμβος

Ο ΔΚ αποτελείται από έναν ηλεκτρονικό υπολογιστή ο οποίος μπορεί να επικοινωνήσει ασύρματα με τους ΕΚ του δικτύου. Ρόλος του ΔΚ είναι να δημιουργήσει και να εκπαιδεύσει το νευρωνικό δίκτυο του οποίου τα συναπτικά βάρη θα διανείμει στους ΕΚ του δικτύου. Το νευρωνικό δίκτυο υλοποιείται κατά την ενεργοποίηση του ΔΚ και στην συνέχεια εκπαιδεύεται, επαληθεύεται και αποθηκεύονται τα συναπτικά βάρη του δικτύου. Κάθε φορά που κάποιος κόμβος του δικτύου αναλαμβάνει το ρόλο του ΕΚ του cluster στο οποίο ανήκει, αποστέλλει αίτημα απόκτησης συναπτικών βαρών στον ΔΚ ο οποίος αποστέλλει τα συναπτικά βάρη του δικτύου που έχει εκπαιδεύσει. Περιοδικά, ο ΔΚ λαμβάνει τις μετρήσεις και τις μέγιστες/ελαχιστες τιμές που έχει ο κάθε ΕΚ. Στο σημείο αυτό, ο ΔΚ για κάθε cluster εκτελεί τις παρακάτω ενέργειες

1. Συγκρίνει τις μέγιστες/ελαχιστες τιμές θερμοκρασίας του κάθε ΑΚ που ανήκει στο συγκεκριμένο cluster, με σκοπό να αναπροσαρμόσει την τιμή κατωφλίου θερμοκρασίας του cluster

2. Συγκρίνει την τυπική απόκλιση της θερμοκρασίας του κάθε ΑΚ, με σκοπό την αναπροσαρμογή τις τιμής κατωφλίου της τυπικής απόκλισης θερμοκρασίας του cluster.
3. Προσθέτει διαρκώς τις μετρήσεις που λαμβάνει, στο υπάρχον Dataset εκπαίδευσης και κάθε 24 ώρες επανεκπαιδεύει το νευρωνικό δίκτυο με το ανανεωμένο Dataset. Οι καινούργιες μετρήσεις που προστίθενται στο υπάρχον Dataset δεν περιέχουν το αποτέλεσμα (0 η 1 ανάλογα με το αν υπάρχει πυρκαγιά η όχι) επομένως η προσθήκη των μετρήσεων χωρίς το αποτέλεσμα τους, καθιστά το υπάρχον Dataset μη αξιοποιήσιμο από το νευρωνικό δίκτυο. Για τον λόγο αυτό, τις μετρήσεις που λαμβάνει από κάθε cluster, τις αποθηκεύει πρώτα σε ένα καινούργιο προσωρινό Dataset. Παράλληλα, από τις μετρήσεις υπολογίζει τον ρυθμό μεταβολής για κάθε μέγεθος και την προκύπτουσα πληροφορία την αποθηκεύει και αυτή στο προσωρινό Dataset. Στην συνέχεια το προσωρινό Dataset τροφοδοτείται σε ένα αλγοριθμο μηχανικης εκμαθησης μη εποπτευομενης εκπαιδευσης και η λειτουργία του είναι να κατηγοριοποιήσει τις καινούργιες μετρήσεις στις δυο κατηγορίες που μας ενδιαφέρουν (ύπαρξη - μη ύπαρξη πυρκαγιάς) χωρίς ανθρώπινη παρέμβαση. Συγκεκριμένα ο αλγοριθμος - για παράδειγμα Self Organizing Map (SOM), λαμβάνει ως είσοδο, τιμή θερμοκρασίας, τιμή υγρασίας, τον ρυθμό μεταβολής της θερμοκρασίας και τον ρυθμό μεταβολής της υγρασίας και εντάσσει την πλειάδα μετρήσεων είτε στην κλάση ύπαρξης πυρκαγιάς είτε στην κλάση μη ύπαρξης πυρκαγιάς[3]. Με αυτό τον τρόπο κάθε ζεύγος μετρήσεων αποθηκεύεται στο αρχικό Dataset με το αναμενόμενο αποτέλεσμα για το συγκεκριμένο ζεύγος μετρήσεων. Κάθε 24 ώρες το νευρωνικό δίκτυο που υλοποιείται και στους ΑΚ επανεκπαιδεύεται με το εμπλουτισμένο Dataset. Εάν η τελευταία εκπαίδευση έχει ως αποτέλεσμα μικρότερο ποσοστό σφάλματος στην λειτουργία του νευρωνικού δικτύου τότε, ο ΔΚ αποστέλλει εκ νέου σε όλους τους ΕΚ τα καινούργια συναπτικά βάρη τα οποία θα συνεχίσει να

[3] Στα πλαίσια της παρούσας εργασίας πραγματοποιήθηκε πειραματική υλοποίηση κατά την οποία νευρωνικό δίκτυο κλήθηκε να κατηγοριοποιήσει τα δεδομένα του υπάρχοντος Dataset στις δυο κλάσεις ενδιαφέροντος χρησιμοποιώντας τον αλγόριθμο SOM. Παρατηρήθηκε πως για τα υπάρχοντα δεδομένα που αποτελούνται από μετρήσεις υγρασίας και θερμοκρασίας, το νευρωνικό δίκτυο είναι σε θέση να κατηγοριοποιήσει δεδομένα που ανήκουν καθαρά είτε στην μια περίπτωση είτε στην άλλη. Για παράδειγμα οι μετρήσεις 67% υγρασία και 15°C(μη ύπαρξη πυρκαγιάς) και 23% υγρασία και 43°C (ύπαρξη πυρκαγιάς) κατηγοριοποιούνται στις ανάλογες κλάσεις. Αντίθετα αδυνατεί να κατηγοριοποιήσει μετρήσεις που ανήκουν στα άκρα της κάθε κατηγορίας. Για παράδειγμα μετρήσεις όπως 19% υγρασία και 23°C (μη ύπαρξη φωτιάς - ξερός ατμοσφαιρικός αέρας) κατηγοριοποιούνται στην κλάση ύπαρξης πυρκαγιάς. Για το λόγο αυτό κρίθηκε απαραίτητη η προσθήκη παραπάνω διαστάσεων στις μετρήσεις. Πρέπει να αναφερθεί πως πιθανώς άλλοι αλγόριθμοι Ενισχυτικής Εκμάθησης όπως Actor-Critic ή μη επιβλεπομένης εκπαίδευσης όπως kmeans Clustering, αποτελούν καλύτερες λύσεις για το παρών πρόβλημα.

αποστέλλει και σε όλους τους καινούργιους ΕΚ που εισέρχονται στο δίκτυο μέχρι αυτά τα συναπτικά βάρη να αντικατασταθούν στον ΔΚ από κάποια νέα βάρη που θα προκύψουν από μελλοντική επεξεργασία μετρήσεων.

5.2. Συμπεράσματα

Το βασικό ερώτημα στο οποίο καλείται να απαντήσει η παρούσα εργασία είναι εάν μπορεί να υλοποιηθεί ένα νευρωνικό δίκτυο σε ένα ασύρματο δίκτυο αισθητήρων υλοποιημένο στην πλατφόρμα SunSPOT της Oracle και εάν είναι εφικτό, πόσο αποτελεσματικό μπορεί να είναι το νευρωνικό δίκτυο. Οι δοκιμές που έγιναν κατά τη διάρκεια της εργασίας υποδεικνύουν ότι είναι εφικτή η ανάπτυξη νευρωνικού δικτύου σε ένα ασύρματο δίκτυο αισθητήρων και τα αποτελέσματα των δοκιμαστικών επαναλήψεων έδειξαν ότι το ποσοστό σφάλματος μπορεί να περιοριστεί σε πολύ μικρό ποσοστό - της τάξης του 5% - εάν τα δεδομένα εκπαίδευσης του νευρωνικού δικτύου καλύπτουν όλο το εύρος μετρήσεων τόσο σε κατάσταση πυρκαγιάς όσο και ηρεμίας. Φανερά μειονεκτήματα στην παρούσα εργασία είναι η περιορισμένη φύση της εισόδου (αποτελείται μόνο από μετρήσεις θερμοκρασίας - υγρασίας) του νευρωνικού δικτύου καθώς και η έλλειψη μετρήσεων σε συνθήκες πραγματικής πυρκαγιάς. Ως έχει το σύστημα είναι σε θέση να ανιχνεύσει πυρκαγιά μόνο εφόσον κάποια από τις μετρήσεις, είτε αυτή της θερμοκρασίας, είτε αυτή της υγρασίας, περάσει την τιμή κατωφλίου που έχει τεθεί γι αυτή τη μέτρηση. Χρησιμοποιώντας για την εκπαίδευση, δεδομένα που περιέχουν πραγματικές μετρήσεις υπό συνθήκες πυρκαγιάς, η είσοδος του νευρωνικού δικτύου θα μπορούσε να αποτελείται εκτός από το ζεύγος μετρήσεων και από το ρυθμό μεταβολής της κάθε μέτρησης του ζεύγους, καθιστώντας την έξοδο του νευρωνικού δικτύου περισσότερο εκφραστική. Επιπλέον πληροφορίες όπως ο μήνας του χρόνου ή η ώρα κατά την οποία λαμβάνεται η μέτρηση η ακόμα και ο δείκτης Fire Weather Index (FWI), μπορούν να προσδώσουν ακόμα μεγαλύτερη λειτουργικότητα στο νευρωνικό δίκτυο καθιστώντας το ικανό να ανιχνεύσει μία πυρκαγιά πριν κάποια από τις τιμές είτε της υγρασίας είτε της θερμοκρασίας περάσει κάποιο σαφώς ορισμένο όριο.

Συνοψίζοντας, το νευρωνικό δίκτυο αν και εξαιρετικά περιορισμένο εξαιτίας της απουσίας βιβλιοθήκης σε γλώσσα που υποστηρίζεται από την πλατφόρμα της Oracle, υλοποιήθηκε επιτυχώς. Τα αποτελέσματα του νευρωνικού δικτύου είναι αρκετά εκφραστικά και δίνουν μία καλή διαίσθηση για την κατάσταση που επικρατεί στον περιβάλλοντα χώρο ενώ παράλληλα χαρακτηρίζονται από μικρό ποσοστό λάθους. Συνεπώς μπορούμε να πούμε ότι αποτελεί

εφαρμόσιμη λύση για ανίχνευση πυρκαγιάς και παρακολούθηση περιβαλλοντικών μεγεθών ενώ παράλληλα έχει μεγάλο περιθώριο βελτίωσης.

Παράρτημα

Πίνακας Ορολογίας

Όρος		Εξήγηση
Java**SE** (**S**tandard **E**dition)	Java Standard Edition	Η τυπική έκδοση της γλώσσας Java. Περιέχει όλες τις βασικές βιβλιοθήκες της γλώσσας (.lang, .io, .math, .net, .util)
Java**ME** (j2me) **M**icro **E**dition	Java Micro Edition	Η έκδοση Micro της γλώσσας Java αποτελεί την πλατφόρμα ανάπτυξης εφαρμογών για κινητές/φορητές συσκευές και embedded συστήματα
Regression	Οπισθοδρόμηση	Τρόπος λειτουργίας νευρωνικού δικτύου κατά τον οποίο το νευρωνικό δίκτυο καλείται να δώσει μια αριθμητική εκτίμηση για το υπό εξέταση μέγεθος
Classification	-	Τρόπος λειτουργίας νευρωνικού δικτύου κατά τον οποίο το νευρωνικό δίκτυο καλείται να κατηγοριοποιήσει τα δεδομένα σε κλάσεις
CLDC	Connected Limited Device Configuration	Είναι μια διασάφηση (specification) ενός framework της JavaME, η οποία περιγράφει ένα σύνολο βιβλιοθηκών καθώς και κάποια χαρακτηριστικά του Virtual Machine που πρέπει να υπάρχουν σε μια υλοποίηση για φορητές συσκευές/embedded συστήματα
MIDP	Mobile Information Device Profile	Σύνολο από API's που καθορίζουν πως μια εφαρμογή αλληλεπιδρά με συσκευές κινητών τηλεφώνων και φορητές συσκευές
Midlet		Εφαρμογή που χρησιμοποιεί τα CLDC και MIDP και προορίζεται για συσκευή κινητού τηλεφώνου η φορητή συσκευή
Dataset	Σύνολο Δεδομένων	Είναι το σύνολο δεδομένων που προορίζεται είτε για εκπαίδευση είτε για επαλήθευση ενός νευρωνικού δικτύου
Training Error (trainError)	Σφάλμα Εκπαίδευσης	Το μέγιστο επιτρεπτό σφάλμα κατά την εκπαίδευση νευρωνικού δικτύου. Ορίζεται από τον σχεδιαστή. Ένα νευρωνικό δίκτυο παραμένει σε κατάσταση εκπαίδευσης όσο το ποσοστό σφάλματος εξόδου του είναι

		μεγαλύτερο από το σφάλμα εκπαίδευσης
Validation Error	Σφάλμα επαλήθευσης	Το ποσοστό σφάλματος εξόδου του δικτύου σε Σετ Δεδομένων - με γνωστή έξοδο - που δεν έχει χρησιμοποιηθεί στην εκπαίδευση
Epoch	-	Η μονάδα μέτρησης των φορών που ολόκληρο το σετ δεδομένων εκπαίδευσης έχει τροφοδοτηθεί στο νευρωνικό δίκτυο
Reservoir Sampling	Δειγματοληψία Δεξαμενής	Μέθοδος δειγματοληψίας για τυχαία επιλογή δείγματος, από μια λίστα δεδομένων
Moving Average	Κινούμενος Μέσος Όρος	Στατιστικό εργαλείο, για επιβεβαίωση τάσεων σε ροές δεδομένων
Basestation	Σταθμός Βάσης	SPOT το οποίο είναι σχεδιασμένο έτσι ώστε να μπορεί να συνδεθεί σε ηλεκτρονικό υπολογιστή και να λειτουργεί ως σύνδεσμος ανάμεσα σε άλλα SPOT (Free Range SPOT)
Aggregator	Συσσωρευτής	SPOT του οποίου η λειτουργία είναι να λαμβάνει, να αποθηκεύει μετρήσεις από Sampler SPOT και να τροφοδοτεί με αυτές το νευρωνικό δίκτυο που υλοποιεί.
Sampler	Δειγματολήπτης	SPOT του οποίου λειτουργία είναι να λαμβάνει μετρήσεις από το περιβάλλον και να τις αποστέλλει στο Aggregator SPOT

Αναφορές.

[1] Abu Alsheikh, Lin, Niyato, Tan. Machine Learning in Wireless Sensor Networks: Algorithms, Strategies and Applications. 2014/5/18

[2] Dr. Robert Hecht-Nielsen In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

[3] NeuralSolutions - Artificial Neural Network Analogy to Human Brain

[4] Heaton Programming Neural Networks in Java.

[5]. UToronto.ca - Depiction of Neural Network Analogy to Human Brain

[6].StackExchange - Hidden Layer Importance in Artificial Neural Networks

[7]. Encog3Java-User. - Chapter 2.2.1 Normalizing Numerical Values

[8]. StackOverflow - Role of a Bias Neuron to an Artificial Neural Network

[9] Neural Network Calculation: Feedforward Neural Network Calculation

[10]. Heaton, J. (2015). Encog: Library of Interchangeable Machine Learning Models for Java and C#. *Journal of Machine Learning Research*, 16, 1243-1247 [10].

[11]. GitHub - Encog Library. Github Repository

[12]. GitHub - EncogME. Java Micro Edition Artificial Neural Network

[13]. Gregable.com - Majority Voting

[14]. SourceForge.net - Math Library for JavaME - REAL.java

[15]. StackOverflow - Artificial Neural Network Output As a Probability

[16]. GeeksforGeeks - Reservoir Sampling Definition

[17]. Gregable - Reservoir Sampling Explanation

[18]. Wikipedia - Moving Average. Definition

[19]. Investopedia - Moving Average. Explanation

[20]. Investopedia - Moving Average. Usage

[21] Liyang Yu, Neng Wang, Xiaoqiao Meng. Real time Forest Fire Detection with Wireless Sensor Networks.

[22] http://sensorapp.net/

[23]. Youssed Safi, Abdelaziz Bouroumi Prediction of Forest Fires Using Artificial Neural Networks

[24]. Arnoldo Diaz-Ramirez, Luis A.Tafoya, Jorge A.Atempa, Pedro Mejia-Alvarez. Wireless Sensor Networks and Fusion Information Methods for Forest Fire Detection

[25]. Yong Poh Yu1 , Rosli Omar1 , Rhett D. Harrison2 , Mohan Kumar Sammathuria3 and Abdul Rahim Nik4. Pattern clustering of forest fires based on meteorological variables and its classification using hybrid data mining methods