Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Υλοποίηση μιας Περιορισμένης Μηχανής Boltzmann με την τεχνική FPGA-in-the-Loop** **Implementation of a Restricted Boltzmann Machine using FPGA-in-the-Loop technique**
Ονοματεπώνυμο Φοιτητή	**Πασχάλης Δημήτριος του Παναγιώτη**
Αριθμός Μητρώου	**ΜΠΣΠ12058**
Κατεύθυνση	**Τεχνολογία Ενσωματωμένων Υπολογιστικών Συστημάτων**
Επιβλέπων	**Μιχάλης Ψαράκης, Επίκουρος Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Μ. Ψαράκης (επιβλέπων) Επίκουρος Καθηγητής	Χ. Κωνσταντόπουλος Επίκουρος Καθηγητής	Α. Πικράκης Επίκουρος Καθηγητής

Περιεχόμενα

Πίνακας εικόνων και γραφημάτων

Πρόλογος

Ένας εκ των τομέων της πληροφορικής που γνωρίζει μεγάλη άνθηση τα τελευταία χρόνια είναι αυτός της Μηχανικής Μάθησης (Machine Learning). Ο τομέας αυτός αναπτύσσεται ραγδαία και ενισχύεται με καινούργιες τεχνικές και τεχνολογίες. Με την ανάπτυξη καλύτερων αλγορίθμων ξεπερνιούνται παλιότερες δυσκολίες και πλέον καθίσταται δυνατή η επίλυση πολύπλοκων προβλημάτων που στο παρελθόν φαίνονταν αδύνατα. Ένας τέτοιος αλγόριθμος είναι αυτός των Περιορισμένων Μηχανών Boltzmann (Restricted Boltzmann Machines),τα οποία είναι ένα είδος Τεχνητού Νευρωνικού Δικτύου. Τα δίκτυα αυτά χρησιμοποιούνται σε ακόμα μεγαλύτερα νευρωνικά δίκτυα σχηματίζοντας τα λεγόμενα Δίκτυα Βαθιάς Πεποιθήσεως (Deep Belief Networks) που αποτελούν την καλύτερη τεχνική για επίλυση δύσκολων προβλημάτων Μηχανικής Μάθησης. Ο κύριος τρόπος υλοποίησης τους είναι το περιβάλλον Matlab και άλλα τέτοιου τύπου λογισμικά περιβάλλοντα. Αυτού του είδους τα δίκτυα όμως είναι απαιτητικά σε πόρους και χρειάζεται συνήθως μεγάλος χρόνος για να εκπαιδευτούν σωστά, κάτι που απαγορεύει την άμεση χρησιμοποίησή τους σε εμπορικές ή βιομηχανικές εφαρμογές. Για αυτό πλέον στο πεδίο έχουν εισέλθει και ερευνητές από το πεδίο της Ψηφιακής Σχεδίασης και Αρχιτεκτονικής Η/Υ,όπου σε συνεργασία με ερευνητές από το πεδίο των Τεχνητών Νευρωνικών Δικτύων, προσπαθούν να σχεδιάσουν υλικό που να εκτελεί τέτοιου είδους υπολογισμών προσπαθώντας να επιτύχουν καλύτερη απόδοση σε καλύτερους χρόνους. Έτσι έχουμε πλέον σχεδιάσεις σε FPGAs και GPUs που, με τους όποιους περιορισμούς, σημειώνουν αξιόλογη πρόοδο ανοίγοντας το δρόμο για επιπλέον ερευνητές που ακολουθούν.

Τα FPGAs (Field Programmable Gate Arrays) είναι ολοκληρωμένα κυκλώματα γενικής χρήσης ικανά για προγραμματισμό. Έχουν έμφυτη την έννοια της παραλληλίας και συχνά χρησιμοποιούνται ως μέσο επιτάχυνσης εφαρμογών. Κομμάτι υπολογισμών, και ειδικότερα παράλληλων, μπορεί να μεταφερθεί σε επίπεδο υλικού προσφέροντας καλύτερους χρόνους εκτέλεσης (επιτάχυνση υλικού). Έτσι έχουμε την δημιουργία εφαρμογών που μοιράζουν τους υπολογισμούς μεταξύ γενικού σκοπού επεξεργαστών (Software) και FPGA (Hardware) με σκοπό την μέγιστη απόδοση. Μια τεχνική για την επίτευξη τέτοιου είδους εφαρμογών αποτελεί και η τεχνική FPGA In the Loop. Σκοπός της παρούσης εργασίας είναι μια εναλλακτική πρόταση για την εκτέλεση λειτουργιών μιας Restricted Boltzmann Machine με χρησιμοποίηση Matlab Simulink και της τεχνικής FPGA In the Loop για επίτευξη Software - Hardware Co-Simulation. Στοχεύουμε στην δημιουργία ενός πλαισίου, με χρήση σχεδιαστικών τεχνικών, που θα καθιστά δυνατή την μεταφορά υπολογισμών σε επίπεδο υλικού, και που θα μπορεί να χρησιμοποιείται με διαφορετικές παραμέτρους ανάλογα με την εκάστοτε σχεδίαση ενός Restricted Boltzmann Machine.

Abstract

One of the fields of computer science that flourishes these last years is that of Machine Learning. The field grows rapidly and is enhanced with new techniques and technologies. By developing stronger algorithms, past difficulties are overcome and is now possible to solve complicated problems that previously seemed impossible. One such algorithm is that of Restricted Boltzmann Machines, which are a type of Artificial Neural Network. Such networks are used to build even greater neural networks, the so called Deep Belief Networks, which provide the best technique for solving difficult Machine Learning problems. They are mainly developed in Matlab environment and other such kinds of software. These kinds of networks, though, are highly demanding in resources and usually require a great amount of time for proper training. That is the main reason that prohibits their use for immediate commercial or industrial applications. As a result the field has been enriched with researchers from the fields of Digital Design and Computer Architecture that cooperate with researchers from the field of Artificial Neural Networks trying to design hardware, capable of executing such calculations and achieve better efficiency as well as reducing the execution time. Furthermore FPGA and GPU designs

have been proposed that, despite of some restrictions, have made noteworthy progress paving the way for more researchers to follow.

FPGAs (Field Programmable Gate Arrays) are generall purpose integrated circuits capable of programming. Parallelism in FPGAs is inherent and they are oftenly used as a means to speed up applications. Part of computations, specifically of parallel nature, can be transferred to hardware level providing better execution time (hardware acceleration). This leads to applications that share computations between general purpose processors (Software) and FPGA (Hardware) for maximum efficiency. On such technique for this kind of applications is FPGA In the Loop. The purpose of this project is to propose an alternative way for executing the functions of a Restricted Boltzmann Machine, by using Matlab Simulink and FPGA In the Loop technique and achieve Hardware Software Co-Simulation. We aim to create a framework, by the use of design techniques, which makes the transfer of the calculations to the hardware level possible and can be used with different parameters according to each specific Restricted Boltzmann Machine design.

1.Εισαγωγή

Οι Περιορισμένες Μηχανές Boltzmann (Restricted Boltzmann Machines – RBM) είναι γενετικά στοχαστικά μοντέλα που αναπαριστούνται ως νευρωνικά δίκτυα. Μπορούν να μοντελοποιήσουν την συμπεριφορά των δεδομένων που εφαρμόζονται στην είσοδό τους, βασιζόμενα σε μια κατανομή δειγμάτων από το συγκεκριμένο σύνολο δεδομένων. Έτσι δημιουργούν ένα εσωτερικό μοντέλο ικανό να αναγνωρίσει δείγματα από αυτή την κατανομή καθώς επίσης και να παράγει δεδομένα που ακολουθούν την συγκεκριμένη κατανομή. Ο όρος στοχαστικά αναφέρεται στον τρόπο με τον οποίο το πετυχαίνει αυτό καθώς χρησιμοποιείται μια πιθανολογική προσέγγιση και το δίκτυο λειτουργεί με βάση την κατανομή πιθανότητας στο σύνολο των δεδομένων που περιγράφουν το πρόβλημα. Αποτελούνται από δύο επίπεδα νευρώνων και δεν υπάρχουν συνδέσεις μεταξύ των νευρώνων που βρίσκονται στο ίδιο επίπεδο.

Χρησιμοποιούνται για την επίλυση προβλημάτων Μηχανικής Μάθησης και είναι ικανά για μάθηση χωρίς επιτήρηση (unsupervised learning). Προσπαθούν να «μάθουν» τις τιμές των παραμέτρων που περιγράφουν καλύτερα την κατανομή των δεδομένων που εφαρμόζονται στην είσοδό τους. Αυτό επιτυγχάνεται μέσω συνεχών εναλλαγών μεταξύ των τιμών των κόμβων των επιπέδων του. Σκοπός είναι , μετά την διαδικασία της εκπαίδευσης, οι τιμές που περιέχονται στις συνδέσεις μεταξύ των κόμβων να περιγράφουν όσο το δυνατόν καλύτερα το μοντέλο.

Ο λόγος που καθιστά τα RBMs τόσο σημαντικά είναι ότι αποτελούν την βάση για πολυπλοκότερα δίκτυα που πλέον χρησιμοποιούνται ευρέως σε δύσκολα προβλήματα Μηχανικής Μάθησης. Αυτά είναι τα λεγόμενα Δίκτυα Βαθιάς Πεποιθήσεως (Deep Belief Networks -DBN). Τέτοιου είδους δίκτυα μπορούν να μάθουν αλληλοεξαρτήσεις μεταξύ αταξινόμητων δεδομένων. Μπορούν να εξάγουν χαρακτηριστικά για προβλήματα που δεν είναι ξεκάθαρα ορισμένα, και να μας οδηγήσουν στην κατηγοριοποίηση και πρόβλεψη δεδομένων για τα προβλήματα αυτά.

Τα DBN έχουν αρκετά επίπεδα νευρώνων και αποτελούνται από RBM που έχουν πολύ μεγάλο αριθμό κόμβων. Αυτό κάνει την όλη διαδικασία εκπαίδευσης αρκετά χρονοβόρα και καθιστά τα DBN μη εφαρμόσιμα άμεσα σε real time βιομηχανικές ή εμπορικές εφαρμογές. Το πρόβλημα αυτό προσπαθούν να ξεπεράσουν ερευνητές από το πεδίο της Ψηφιακής Σχεδίασης και Αρχιτεκτονικής Η/Υ, καθώς γίνονται προσπάθειες να μεταφερθούν οι υπολογισμοί που λαμβάνουν μέρος, σε επίπεδο Υλικού.

Όπως αναφέραμε στον πρόλογο τα FPGAs χρησιμοποιούνται πλέον ευρέως ως επιταχυντές εφαρμογών καθώς μπορούν να σχεδιαστούν ,σε αυτά, δομές hardware που εκτελούν γρήγορους υπολογισμούς. Διαθέτουν πλήθος λογικών στοιχείων που μπορούν να συνδυαστούν για την εκτέλεση οποιουδήποτε υπολογισμού. Τα στοιχεία αυτά μπορούν να λειτουργούν παράλληλα κάτι που κάνει τα FPGAs ιδανικά για επίλυση προβλημάτων που έχουν έμφυτη την έννοια της παραλληλίας. Στα RBM, λόγω του περιορισμού των συνδέσεων μεταξύ των

νευρώνων, οι υπολογισμοί που λαμβάνουν μέρος μεταξύ των δύο επιπέδων μπορούν να γίνουν παράλληλα. Έτσι έχουν γίνει αρκετές αξιόλογες προσπάθειες για την μεταφορά των λειτουργιών μιας RBM σε FPGA. Τα προβλήματα που εμφανίζονται σε αυτές τις προσπάθειες έχουν να κάνουν με το μέγεθος των δικτύων που μπορούν να εξυπηρετήσουν, την επεκτασιμότητα αυτών, την παραμετροποίηση για μια γενικότερη περίπτωση και τέλος την εξειδικευμένη γνώση που πρέπει να έχει κάποιος για καταφέρει μια τέτοια υλοποίηση σε επίπεδο υλικού.

Μέσω της παρούσης εργασίας γίνεται μια προσπάθεια επίλυσης των προηγούμενων προβλημάτων και μια εναλλακτική πρόταση για την υλοποίηση μιας RBM με χρήση τόσο software για την παραμετροποίηση ,σχεδίαση και διαχείριση των δεδομένων μιας RBM όσο και χρήση FPGA για την επιτάχυνση των υπολογισμών. Αυτό γίνεται με την τεχνική FPGA In the Loop (FIL) και την σχεδίαση των λειτουργιών μιας RBM με την χρήση του εργαλείου Matlab Simulink. Εξερευνούμε τα πλεονεκτήματα και μειονεκτήματα μιας τέτοιας σχεδίασης έχοντας ως γνώμονα ότι είναι ικανή για εύκολη παραμετροποίηση και δεν χρειάζεται εξειδικευμένη γνώση hardware. Μέσω της FIL σχεδίασης μοιράζουμε τους υπολογισμούς σε software και σε hardware καθώς αναθέτουμε τις διαδικασίες που επιθυμούμε είτε στο Matlab Simulink είτε στο FPGA ανάλογα με το τι θέλουμε να πετύχουμε.

Όπως προείπαμε σκοπός της παρούσης εργασίας είναι η δημιουργία ενός Software – Hardware μοντέλου ικανού να εκτελεί τις λειτουργίες μιας RBM. Πριν προχωρήσουμε όμως στην παρουσίαση αυτού του μοντέλου θα πρέπει πρώτα να κατανοήσουμε έννοιες που αφορούν σε ένα πλήθος όρων και διαδικασιών από διαφορετικά πεδία της επιστήμης της πληροφορικής, καθώς επίσης και να κατανοήσουμε τους λόγους που μας οδήγησαν στην ενασχόληση με αυτό το έργο και γιατί θεωρείται πλέον τόσο σημαντικό στην σύγχρονη εξέλιξη της Μηχανικής Μάθησης (Machine Learning - ML). Θα προσπαθήσουμε να δομήσουμε το υπόλοιπο της εργασίας με τρόπο τέτοιο ώστε να είναι όσο το δυνατόν περισσότερο κατανοητές οι έννοιες - διαδικασίες και το πώς περνάμε από το ένα στάδιο στο άλλο προσπαθώντας παράλληλα να αποφύγουμε την υπερβολική ανάλυση της θεωρίας των RBM και των Τεχνητών Νευρωνικών Δικτύων (Artificial Neural Networks - ANN),κάτι που γίνεται εκτενέστερα και καλύτερα τεκμηριωμένα στις εργασίες – πηγές της βιβλιογραφίας. Έτσι στο επόμενο δεύτερο κεφάλαιο θα στοχεύσουμε στο απαραίτητο θεωρητικό υπόβαθρο που αφορά στην Μηχανική Μάθηση, τα προβλήματα που επιλύει και τις εφαρμογές που βρίσκει καθώς επίσης και στα Νευρωνικά Δίκτυα (Neural Networks -NN). Στο τρίτο Κεφάλαιο θα παρουσιάσουμε τα RBM ως Νευρωνικά Δίκτυα,την εκπαίδευση μιας RBM καθώς επίσης και την θεωρία των Deep Belief Networks και θα παρουσιάσουμε κάποια παραδείγματα – χρήσεις που βρίσκουν. Στο κεφάλαιο 4 θα κάνουμε μια σύντομη ανάλυση των προσπαθειών που έχουν γίνει για την μεταφορά RBM σε επίπεδο υλικού, κυρίως σε FPGA, καθώς επίσης στις ιδιαιτερότητες/προβλήματα/περιορισμούς τέτοιων υλοποιήσεων. Στο κεφάλαιο 5 θα εισέλθουμε πλέον σε ότι αφορά την υλοποίηση της παρούσης εργασίας. Θα κάνουμε μια παρουσίαση των λόγων που επιλέξαμε αυτή την προσέγγιση, του κώδικα που επιλέξαμε να σχεδιάσουμε, τον τρόπο λειτουργίας του Matlab Simulink και τις ενέργειες που απαιτούνται για την FPGA In the Loop (FIL) σχεδίαση ενός τέτοιου αλγορίθμου. Επιπλέον θα κάνουμε μια σύντομη ανάλυση της φύσης του dataset που χρησιμοποιούμε και των ιδιαιτεροτήτων/περιορισμών που παρουσιάζονται καθώς και μια σύντομη εισαγωγή στον τρόπο λειτουργίας των FPGA . Τέλος στο κεφάλαιο 6 θα ανακεφαλαιώσουμε, αποτιμώντας την παρούσα προσπάθεια και θα εξάγουμε κάποια χρήσιμα συμπεράσματα.

2.Θεωρητικό υπόβαθρο

2.1 Μηχανική Μάθηση (ML)

Την τελευταία 20ετία ο κλάδος της Μηχανικής Μάθησης έχει κάνει ένα τεράστιο άλμα και πλέον χρησιμοποιείται ευρέως σε αρκετές πλευρές των τεχνολογικών επιτευγμάτων του σήμερα, ακόμη και χωρίς να το «καταλαβαίνουμε» άμεσα. Σε αυτό έχει συντελέσει και ο τεράστιος όγκος δεδομένων που είναι διαθέσιμος [1]. Στην βάση της η Μηχανική Μάθηση είναι μια μέθοδος

ανάλυσης δεδομένων που χρησιμοποιεί αλγόριθμους που «μαθαίνουν» επαναληπτικά από δεδομένα και επιτρέπουν στους υπολογιστές να βρίσκουν κρυφές εξαρτήσεις χωρίς να είναι ειδικά προγραμματισμένοι για αυτό. Επιπλέον χρησιμοποιεί την γνώση αυτών των εξαρτήσεων για να κάνει πρόβλεψη σε δεδομένα ή να πάρει κάποιες αποφάσεις βασισμένη σε αυτά [2]. Αποτελεί κλάδο του πεδίου της Τεχνητής Νοημοσύνης και αφορά αλγορίθμους που βελτιώνουν την εργασία που παράγουν χρησιμοποιώντας την εμπειρία τους και, μαθαίνουν από τα δεδομένα πάνω στα οποία λειτουργούν.

2.1.1 Κατηγορίες Μάθησης

Ανάλογα με τους αλγόριθμους που χρησιμοποιούν μπορούμε να διακρίνουμε τις τρεις ακόλουθες κατηγορίες Μάθησης [2],[3],[4]:

Μάθηση με επιτήρηση (Supervised Learning) ή επιτηρούμενη Μάθηση, κατά την οποία ο αλγόριθμος δέχεται τα παραδείγματα- εισόδους και τα επιθυμητά αποτελέσματα από έναν «δάσκαλο» και στόχος είναι να ανακαλύψει έναν κανόνα που να αντιστοιχεί τις εισόδους με τα αποτελέσματα αυτά.

Μάθηση χωρίς επίβλεψη (Unsupervised Learning) ή μη επιτηρούμενη Μάθηση, κατά την οποία δεν υπάρχει κάποιος «δάσκαλος» - επιτηρητής και σκοπός είναι η εύρεση κανονικοτήτων στις εισόδους. Λειτουργεί είτε ως αυτοσκοπός, να βρεθούν δηλαδή δομές και εξαρτήσεις μεταξύ των δεδομένων εισόδου, είτε ως μέσο για την επίλυση ενός πολυπλοκότερου προβλήματος.

Ενισχυτική Μάθηση (Reinforcement Learning) όπου ένα πρόγραμμα αλληλεπιδρά με ένα δυναμικό περιβάλλον και σκοπός είναι να σχηματιστεί μια πολιτική καλών δράσεων για την καλύτερη επίλυση του προβλήματος. Το πρόγραμμα μαθαίνει από προηγούμενη εμπειρία όπου αξιολογεί παλαιότερες ακολουθίες δράσεων και επιλέγει μια καλή πολιτική.

2.1.2 Εφαρμογές Μηχανικής Μάθησης

Οι εφαρμογές όπου χρησιμοποιείται η Μηχανική Μάθηση, μερικές από τις οποίες παρουσιάζονται παρακάτω[1],[2],[3],[4], δείχνουν γιατί θεωρείται από τα πιο σημαντικά σύγχρονα πεδία της επιστήμης της πληροφορικής.

Αλγόριθμοι Αξιολόγησης Ιστοσελίδων (Web Page Ranking**)** αναφέρεται στον τρόπο που οι μηχανές αναζήτησης βρίσκουν, αξιολογούν και φιλτράρουν ιστοσελίδες με βάση τα όσα εισάγαμε ως είσοδο στο query. Μπορεί να μας φαίνεται ως κάτι σύνηθες και τετριμμένο όμως για να επιτευχθεί μια σωστή αναζήτηση πρέπει να αξιολογηθεί γνώση όπως η δομή μιας ιστοσελίδας,το πόσο σχετική είναι με το query, άλλα παραδείγματα παρόμοιων αναζητήσεων καθώς και να ληφθούν υπόψιν οι βαθμολογίες των επιμέρους ιστοσελίδων.

Συνεργατικό Φιλτράρισμα (Collaborative Filtering) όπου αναφέρεται στον τρόπο που αξιολογούνται οι προαναφερθείσες βαθμολογίες και στις συσχετίσεις που μπορούμε να εξάγουμε από αυτές. Στις σύγχρονες εμπορικές ιστοσελίδες, όπως το Amazon ή το Netflix, χρησιμοποιείται γνώση, όχι συγκεκριμένη με τη μορφή query πλέον, αλλά γενικότερη προερχόμενη καθαρά από προηγούμενες αποφάσεις παρόμοιων χρηστών ώστε να προσφέρονται στους χρήστες στοχευμένες προς το πρόσωπό τους υπηρεσίες – προϊόντα.

Εφαρμογές Ασφάλειας όπως έλεγχος εισόδου με αναγνώριση προσώπου. Δεδομένης μια φωτογραφίας ή ενός βίντεο μπορεί να αναγνωρίζεται κάποιος μέσω αξιολόγησης γνώσης που αφορά διαφορετικές συνθήκες φωτισμού, εκφράσεις προσώπου, κούρεμα και πολλών άλλων χαρακτηριστικών.

Φιλτράρισμα Ενοχλητικής αλληλογραφίας (Spam Filtering) όπου μας ενδιαφέρει να κατηγοριοποιούνται νέα εισερχόμενα μηνύματα, αυτόματα από κάποιο σύστημα που να αξιολογεί προηγούμενη γνώση και να παίρνει απευθείας αποφάσεις για το αν είναι ή δεν είναι spam.

Όπως γίνεται κατανοητό με την χρήση της Μηχανικής Μάθησης μπορούμε να βρούμε συσχετίσεις και εξαρτήσεις δεδομένων και να κάνουμε αποδοτική χρήση αυτών για επίλυση

προβλημάτων που δεν διέπονται από ξεκάθαρα ορισμένους κανόνες. Επιπλέον, αξίζει να αναφέρουμε τις εφαρμογές που βρίσκει η Μηχανική Μάθηση, μεταξύ άλλων,στην Αναγνώριση Φωνής (Windows Vista), Αναγνώριση ψηφίων γραμμένων με το χέρι (PDAs),ανίχνευση σφαλμάτων (π.χ. σε μηχανές jet), στοχευμένη διαφήμιση, αυτόματα ρομπότ καθαρισμού (iRobot Roomba) και φυσικά σε εφαρμογές data mining.

Γενικότερα οι εφαρμογές αυτές εμπίπτουν κατά κύριο λόγο σε προβλήματα **ταξινόμησης** και **κατηγοριοποίησης**. Συμπεραίνουμε λοιπόν ότι δεδομένου ενός μεγάλου όγκου δεδομένων κύρια δύναμη της Μηχανικής Μάθησης είναι η κατηγοριοποίηση των δεδομένων αυτών και η εξαγωγή κανόνων και συσχετίσεων που βοηθούν στην λήψη αποφάσεων και προβλέψεων.

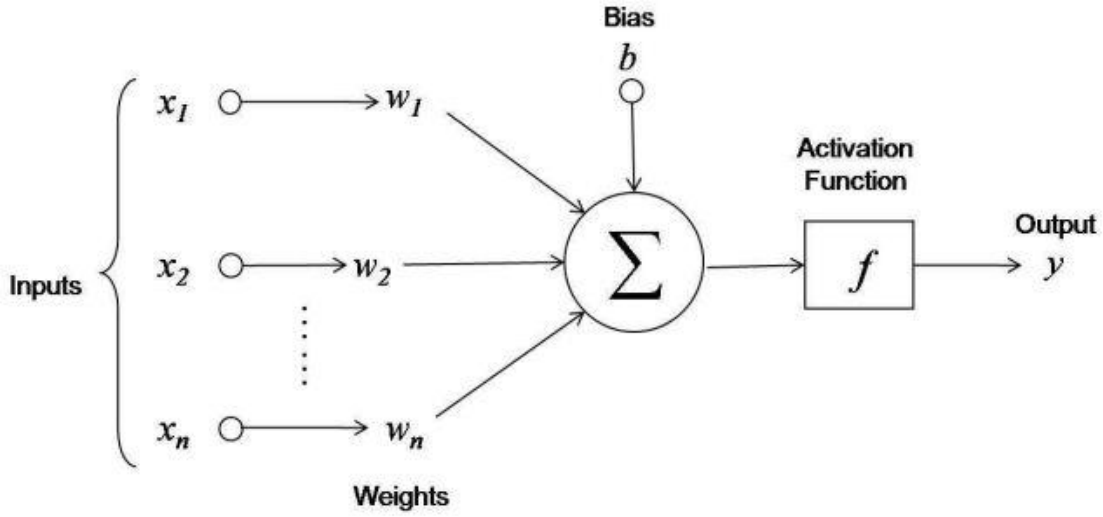
2.2 Τεχνητά Νευρωνικά Δίκτυα

Όπως προαναφέραμε η Μηχανική Μάθηση χρησιμοποιεί διαφόρων ειδών αλγορίθμους για να επιτύχει τις εκάστοτε λειτουργίες που απαιτούνται για την επίλυση ενός προβλήματος. Ανάλογα με το είδος των δεδομένων που διαθέτουμε χρησιμοποιείται και ο κατάλληλος αλγόριθμος. Ένας από τους πιο διαδεδομένους και πλέον χρησιμοποιούμενους στο πεδίο είναι αυτός των Τεχνητών Νευρωνικών Δικτύων. Ίσως να είναι και ο πλέον κατάλληλος με την έννοια ότι όπως η Μηχανική Μάθηση προσομοιώνει τον τρόπο που «μαθαίνει» ένας άνθρωπος έτσι και τα Τεχνητά Νευρωνικά δίκτυα προσπαθούν να προσομοιώσουν τον τρόπο που λειτουργεί ο ανθρώπινος εγκέφαλος[5]. Όπως ο ανθρώπινος εγκέφαλος αποτελείται από ένα τεράστιο δίκτυο βιολογικών νευρώνων, συνδεδεμένους μεταξύ τους έτσι και τα ΤΝΔ αποτελούνται από έναν (μικρότερο) αριθμό τεχνητών νευρώνων συνδεδεμένους κατάλληλα. Στα δύο αυτά είδη δικτύων εμφανίζονται τα εξής αντίστοιχα χαρακτηριστικά [5],[6] :

Ανθρώπινος εγκέφαλος	Τεχνητό Νευρωνικό Δίκτυο
οι νευρώνες έχουν ρυθμιζόμενες παραμέτρους ώστε να διευκολύνεται η διαδικασία της μάθησης - πλαστικότητα του νευρώνα	η γνώση αποκτάται από το δίκτυο μέσα από μια διαδικασία μάθησης - εκπαίδευσης
το δίκτυο αποτελείται από πολλούς νευρώνες ώστε να επιτυγχάνεται παραλληλισμός της επεξεργασίας και διαμοιρασμός της πληροφορίας	η γνώση αποθηκεύεται στις συνδέσεις των νευρώνων, τα λεγόμενα συναπτικά βάρη

2.2.1 Βασικά συστατικά των Τεχνητών Νευρωνικών Δικτύων

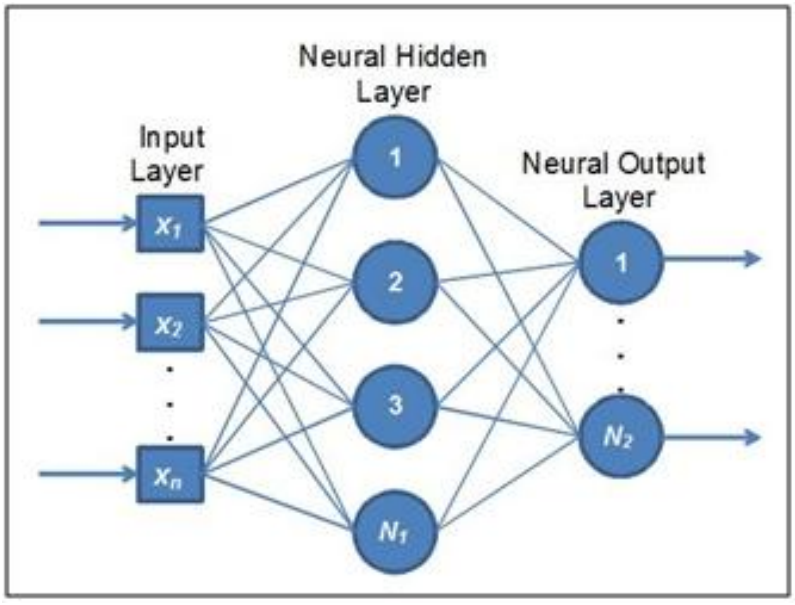
Κύριο συστατικό ενός ΤΝΔ είναι ο Νευρώνας. Αποτελεί την βασική επεξεργαστική μονάδα του συστήματος. Δέχεται εισόδους τις οποίες αξιολογεί μέσα από μια Συνάρτηση Ενεργοποίησης (activation function) ή μια Συνάρτηση Μεταφοράς (transfer function) και παράγει μια έξοδο. Οι είσοδοι αυτοί σε πρώτο επίπεδο είναι τα δεδομένα του προβλήματος και σε επόμενα επίπεδα μπορεί να αποτελούν εξόδους προηγούμενων νευρώνων. Κάθε σύνδεση ενός νευρώνα με έναν άλλο ονομάζεται Σύναψη. Κάθε σύναψη «συνεισφέρει» διαφορετικά στο σύστημα και καθορίζει την αλληλεπίδραση μεταξύ των νευρώνων. Αυτό επιτυγχάνεται με την απόδοση μιας τιμής σε κάθε σύναψη που ονομάζεται Συναπτικό Βάρος (synaptic weight). Στην παρακάτω εικόνα απεικονίζεται η δομή ενός τεχνητού νευρώνα :



Εικόνα 1 Δομή Τεχνητού Νευρώνα

2.2.2 Μορφή - Δομή των Τεχνητών Νευρωνικών Δικτύων

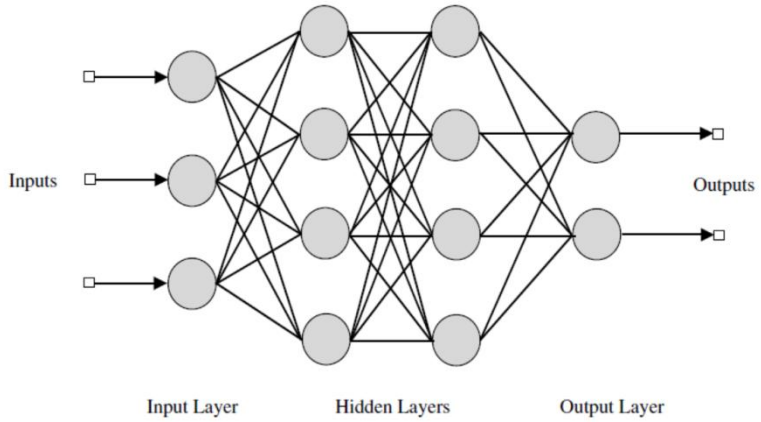
Ένα ΤΝΔ αναπαρίσταται ως ένα γράφημα με κόμβους και συνδέσεις μεταξύ αυτών. Οι κόμβοι είναι στην ουσία οι νευρώνες και οι συνδέσεις αποτελούν τις συνάψεις αυτών. Τα ΤΝΔ οργανώνονται σε επίπεδα και κάθε επίπεδο μπορεί να περιέχει πλήθος νευρώνων. Το πρώτο επίπεδο ενός ΤΝΔ αποτελεί το επίπεδο εισόδου (input layer),όπου γίνεται είσοδος των δεδομένων, και το τελευταίο επίπεδο αποτελεί το επίπεδο εξόδου (output layer) και μας δίνει τα αποτελέσματα. Τα ενδιάμεσα επίπεδα νευρώνων καλούνται και κρυφά επίπεδα (hidden layers). Η πιο απλή μορφή ενός ΤΝΔ απεικονίζεται παρακάτω και αποτελείται από το επίπεδο εισόδου, ένα επίπεδο νευρώνων και το επίπεδο εξόδου.



Εικόνα 2 Απλή μορφή ΤΝΔ

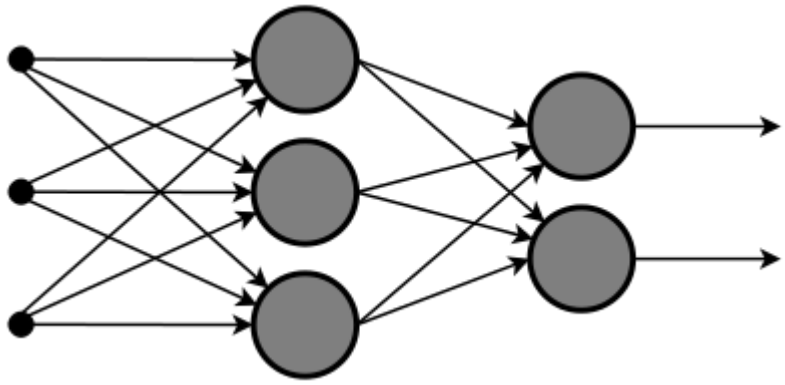
2.2.3 Κατηγορίες Τεχνητών Νευρωνικών Δικτύων

Τα ΤΝΔ στην πραγματικότητα αποτελούνται από πολλά επίπεδα και οι συνδέσεις μεταξύ των νευρώνων μπορεί να διαφοροποιούνται. Η μορφή ενός πολυεπίπεδου ΤΝΔ φαίνεται παρακάτω.

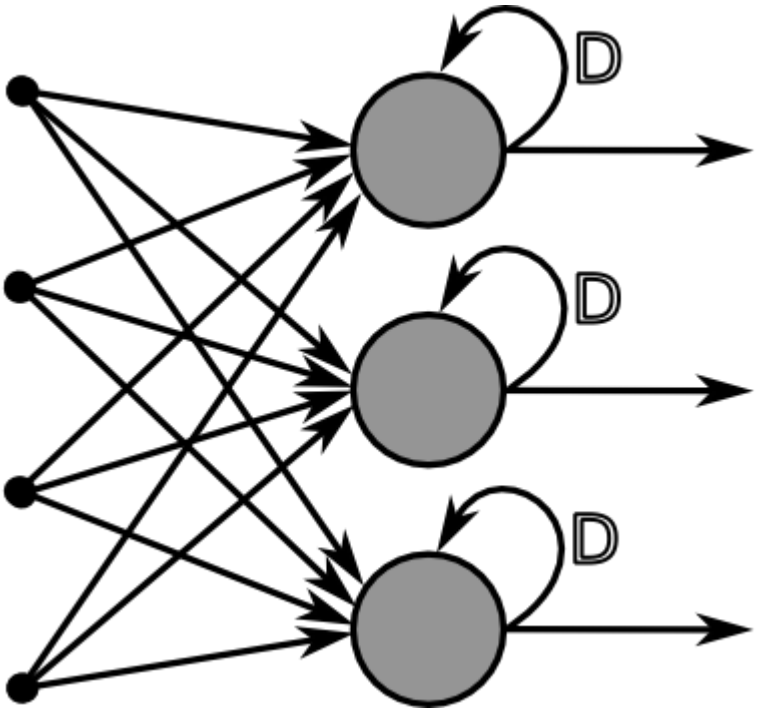


Εικόνα 3 Πολυεπίπεδο ΤΝΔ

Εκτός από τον διαχωρισμό των ΤΝΔ σε δίκτυα ενός επιπέδου και πολυεπίπεδα δίκτυα, μπορούμε να τα ταξινομήσουμε ανάλογα με την μετάδοση των σημάτων σε δύο κύριες κατηγορίες[5][6][7][8]. Στα Εμπρός Τροφοδότησης (Feed Forward) και στα Αναδρομικά (Recurrent) ΤΝΔ. Όπως φαίνεται και από την ονομασία τους στα Feed Forward Networks τα σήματα μεταφέρονται προς μια μόνο κατεύθυνση ενώ στα Recurrent υπάρχει τουλάχιστον ένας βρόγχος ανατροφοδότησης.

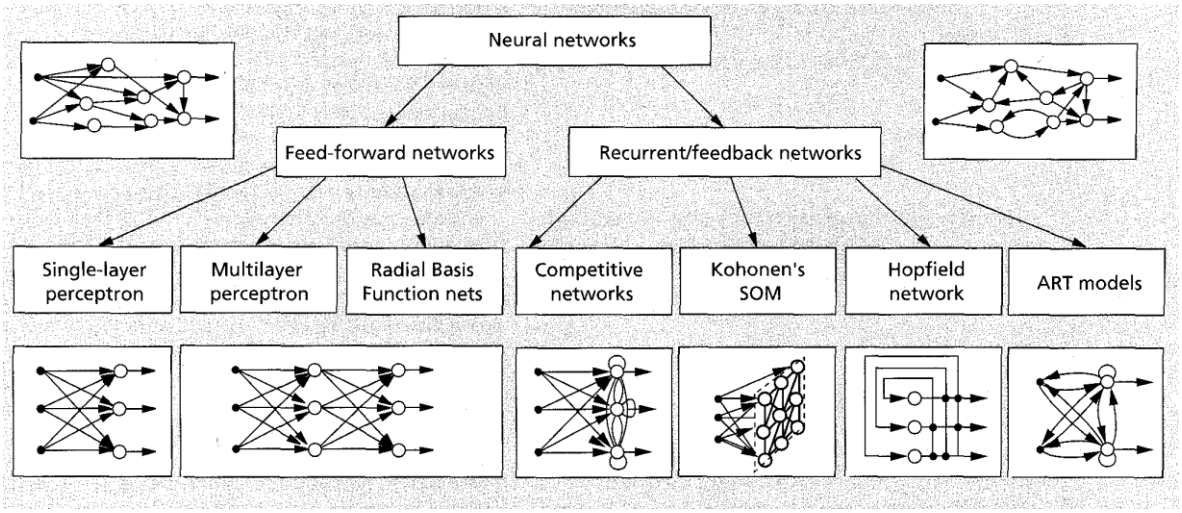


Εικόνα 4 Παράδειγμα Feed Forward Neural Network



Εικόνα 5 Παράδειγμα Recurrent Neural Network

Υπάρχουν επιπλέον παραδείγματα Τεχνητών Νευρωνικών Δικτύων το κάθε ένα από τα οποία επιλύει ποιο συγκεκριμένα προβλήματα[9]. Μια επιπλέον σύντομη απεικόνιση τέτοιων δικτύων φαίνεται στην παρακάτω εικόνα.



Εικόνα 6 Επιπλέον Κατηγοριοποιήσεις Νευρωνικών Δικτύων [9]

Επιπλέον υπάρχει ένας σημαντικός διαχωρισμός που αφορά στα ΤΝΔ και αυτός είναι ο τρόπος με τον οποίο εκπαιδεύονται.

2.2.4 Εκπαίδευση ΤΝΔ

Όπως αναφέραμε στην αρχή,τα ΤΝΔ είναι αλγόριθμοι που επιτελούν μηχανική μάθηση και ως εκ τούτου πρέπει να μπορούν να μαθαίνουν από το περιβάλλον τους και να αξιοποιούν αυτήν την γνώση. Η έννοια της γνώσης στα ΤΝΔ αναφέρεται στην ικανότητα ανακάλυψης και αποθήκευσης των πληροφοριών και αποτελούν εκείνες τις παραμέτρους που ορίζουν ένα κατά τα άλλα ακαθόριστο προς επίλυση πρόβλημα. Στα ΤΝΔ αυτή η γνώση τροποποιείται και αποθηκεύεται στις τιμές των βαρών που φέρουν οι συνάψεις μεταξύ των νευρώνων. Έτσι ανάλογα με τον τρόπο που επιτυγχάνεται αυτό, τα ΤΝΔ διαχωρίζονται με βάση του τρόπου εκπαίδευσης τους [5],[6],[7].

Στην **Μάθηση με Επιτήρηση** χρησιμοποιούνται παραδείγματα του προβλήματος (training data) δίνοντας τα ως είσοδο στο δίκτυό μας. Επιπλέον δίνονται οι επιθυμητές έξοδοι για τα συγκεκριμένα παραδείγματα. Έτσι ξεκινώντας από μια τυχαία αρχική τιμή, που αναθέτουμε στις τιμές των βαρών του δικτύου, αυτό, προσπαθώντας να φτάσει στην επιθυμητή έξοδο για το συγκεκριμένο παράδειγμα, μεταβάλλει επαναληπτικά τις τιμές των βαρών. Αυτό συνήθως επιτυγχάνεται με το να ελέγχεται το σφάλμα της απόκλισης που έχει από την επιθυμητή έξοδο σε κάθε επανάληψη. Όταν έχει ολοκληρωθεί η επαναληπτική διαδικασία το δίκτυο θα έχει καθορίσει τις σωστές τιμές και με βάση αυτές τις παραμέτρους θα μπορεί να παράγει την επιθυμητή έξοδο για κάθε δεδομένο που εφαρμόζεται στην είσοδό του, από το συγκεκριμένο πρόβλημα. Μπορούμε να ελέγξουμε την σωστή λειτουργία του εφαρμόζοντας ένα σύνολο παραδειγμάτων για τα οποία ξέρουμε τις εξόδους (test data).

Στην **Μη Επιτηρούμενη Μάθηση** έχουμε να κάνουμε με πιο δύσκολα προβλήματα. Η Μάθηση με Επιτήρηση λειτουργεί σε καλά ορισμένα προβλήματα όπου γνωρίζουμε τα δεδομένα μας. Στην περίπτωση που έχουμε ελλιπείς ή καθόλου πληροφορίες για τα δεδομένα μας χρησιμοποιούμε την Μη Επιτηρούμενη Μάθηση με σκοπό να βρούμε συσχετίσεις μεταξύ αυτών. Έτσι θα μπορέσουμε για παράδειγμα να τα ταξινομήσουμε και να προβούμε σε επιπλέον ενέργειες επί αυτών. Δεν ζητάμε κάποια συγκεκριμένα αποτελέσματα παρά αφήνουμε τέτοιου είδους αλγορίθμους να τρέξουν κατά ένα ορισμένο διάστημα και με παραμέτρους τέτοιες που πιστεύουμε ότι είναι ικανοποιητικές, για να βρεθούν οι ζητούμενες εξαρτήσεις στα δεδομένα μας.

Τέτοιου είδους αλγόριθμοι χρησιμοποιούνται κατά κόρον σε εφαρμογές Μηχανικής Μάθησης όπου υπάρχει μεγάλος προς εκμετάλλευση όγκος δεδομένων αλλά δεν υπάρχουν άμεσα αντιληπτές κανονικότητες σε αυτά. Παραδείγματα χρήσης τέτοιων αλγορίθμων είναι η αναγνώριση προτύπων, η ταξινόμηση, η συσταδοποίηση (clustering) και παραδείγματα όπως αυτά που προαναφέρθηκαν στην παράγραφο 2.1.2 . Στην πορεία θα συναντήσουμε κάποια επιπλέον παρόμοια παραδείγματα.

Ο αλγόριθμος των Restricted Boltzmann Machines είναι ένας από τους πλέον κατάλληλους αλγόριθμους για χρήση σε τέτοιες περιπτώσεις.

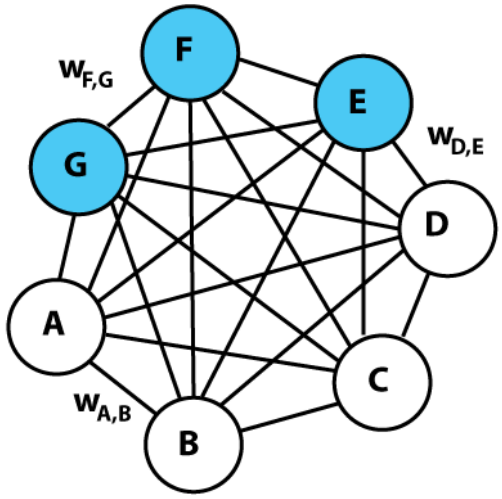
3. Restricted Boltzmann Machines

Οι Restricted Boltzmann μηχανές αποτελούν μια παραλλαγή των απλών μηχανών Boltzmann.

3.1 Boltzmann Machines

Οι μηχανές Boltzmann εφευρέθηκαν από τους Geoffrey Hinton και Terry Sejnowski και ορίζονται ως πιθανολογικά γραφικά μοντέλα που μπορούν να ερμηνευθούν και ως στοχαστικά νευρωνικά δίκτυα[10]. Το κύριο χαρακτηριστικό τους είναι η δυνατότητα ανακάλυψης σημαντικών ιδιοτήτων μιας άγνωστης κατανομής πιθανότητας χρησιμοποιώντας δείγματα από την ίδια την κατανομή[10]. Στον πυρήνα τους είναι ένα δίκτυο συμμετρικά συνδεδεμένων μονάδων – νευρώνων που παίρνουν στοχαστικά αποφάσεις για το αν θα είναι “on” ή “off”[11][12]. Είναι από τα πρώτα δίκτυα που μπορούν να ανακαλύψουν εσωτερικές συσχετίσεις και εξαρτήσεις μεταξύ δεδομένων που εφαρμόζονται στις εισόδους τους και έχουν πάρει το όνομά τους από την

Κατανομή Boltzmann που χρησιμοποιούν κατά την εκτέλεση της συνάρτησης δειγματοληψίας τους[11]. Επιπλέον ανήκουν στα ενεργειακά μοντέλα που σημαίνει ότι όλο το σύστημα έχει μια συμπεριφορά που καθορίζεται από μια τιμή Ενέργειας[13]. Η γραφική αναπαράσταση μιας Boltzmann μηχανής φαίνεται στην παρακάτω εικόνα.



Εικόνα 7 Γραφική αναπαράσταση Boltzmann Machine

Η ενέργεια σε μια BM είναι ίδια σε μορφή με αυτήν ενός δικτύου Hopfield και δίνεται από την σχέση [11] :

$$E = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right)$$

Όπου :

- Wij είναι η τιμή του βάρους μεταξύ της μονάδας j και της μονάδας i
- Si είναι η κατάσταση της μονάδας i με Si ∈ {0,1}
- Θi είναι η προδιάθεση/πόλωση (bias) της μονάδας i, στην συνάρτηση της καθολικής ενέργειας (-θi είναι το κατώφλι ενεργοποίησης της μονάδας)

Οι συνδέσεις σε μια BM έχουν τους εξής δύο περιορισμούς

- Wii = 0 . Καμία μονάδα δεν έχει σύνδεση στον εαυτό της
- Wij = Wji. Όλες οι συνδέσεις είναι συμμετρικές

Η πιθανότητα κάποια μονάδα να είναι “on” δίνεται από την σχέση : $p_{i=on} = \frac{1}{1+ exp\left(-\frac{\Delta E_i}{T}\right)}$ και η κατανομή των ενεργειών των μονάδων ορίζεται από τον νόμο Boltzmann :$P(E) = e^{-\frac{E}{kT}}$
Όπου :

- E είναι η ενέργεια του συστήματος
- P(E) η πιθανότητα το σύστημα να βρίσκεται σε κατάσταση με ενέργεια E
- k η σταθερά Boltzmann
- T η θερμοκρασία

Συμπεραίνουμε ότι όταν η θερμοκρασία παίρνει μεγάλες τιμές, και επειδή το T είναι στον παρανομαστή του κλάσματος, έχουμε $P(E) = 1$. Αυτό για το σύστημα σημαίνει ότι κάθε ενεργειακή κατάσταση είναι πιθανή σαν ενδεχόμενο με πιθανότητα 1.

Αντίστοιχα όταν η θερμοκρασία ελαττώνεται, η πιθανότητα να έχουμε μεγάλη ενέργεια μειώνεται.

Οι αρχές της μεθόδου Boltzmann που εφαρμόζονται στην στατιστική μηχανική μας δίνουν τον τρόπο που εκπαιδεύεται μια μηχανή Boltzmann.

1. Δίνουμε μία αρχική τυχαία μεγάλη τιμή στην παράμετρο T του συστήματος
2. Εφαρμόζουμε στην είσοδο τα πρότυπα μας και υπολογίζουμε την έξοδο και το σφάλμα
3. Κάνουμε μία τυχαία αλλαγή στα βάρη w και υπολογίζουμε εκ νέου την έξοδο
4. Συγκρίνουμε τα σφάλματα και αν το σφάλμα μικραίνει κρατάμε την αλλαγή
5. Εάν το σφάλμα μεγαλώνει τότε υπολογίζουμε την πιθανότητα $P(c) = e^{-\frac{c}{kT}}$ να κρατήσουμε την αλλαγή αυτή

   Όπου
   - k είναι σταθερά, αντίστοιχη προς την σταθερά Boltzmann, και εξαρτάται από το εκάστοτε πρόβλημα,
   - T είναι η «θερμοκρασία» του συστήματος
   - P(c) είναι η πιθανότητα για μία αλλαγή c στη συνάρτηση σφάλματος.
6. Διαλέγουμε έναν τυχαίο αριθμό r, από μία ομοιόμορφη κατανομή $0 < r < 1$. Εάν $P(c) > r$, τότε κρατάμε την αλλαγή. Εάν $P(c) < r$, τότε η αλλαγή απορρίπτεται και προχωράμε σε μία νέα αλλαγή στα βάρη w.

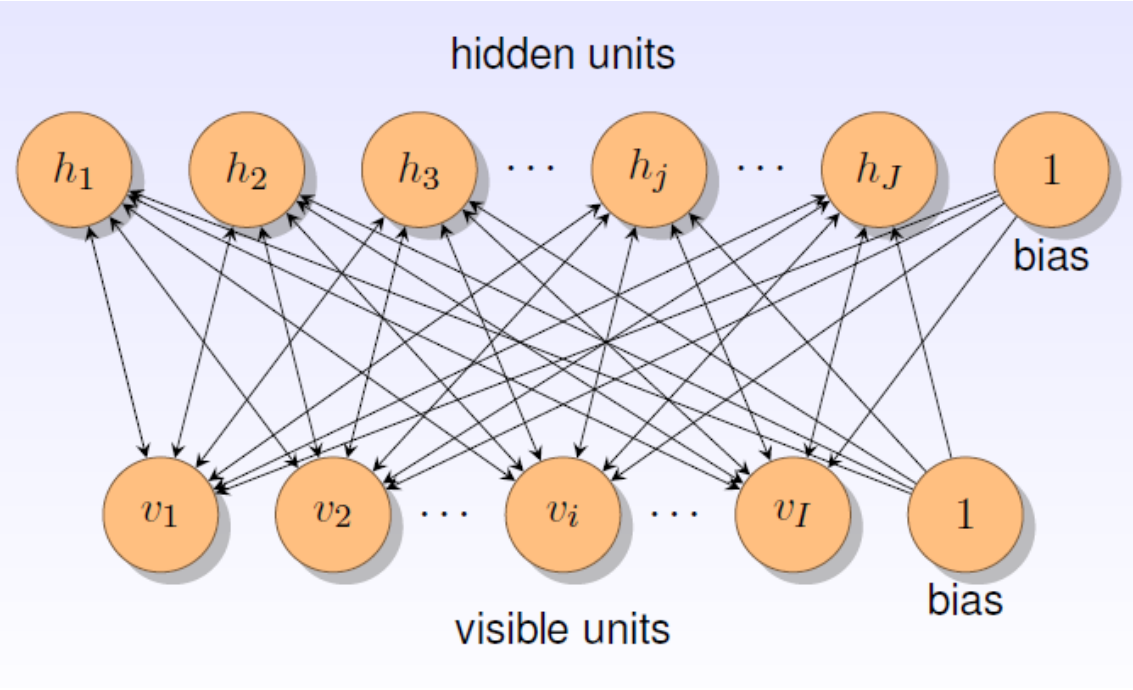
Ήδη έχουμε αρχίσει να παρατηρούμε ομοιότητες με τον τρόπο εκπαίδευσης ενός νευρωνικού δικτύου που αναφέραμε στην παράγραφο 2.2.4 . Θέλοντας να φτάσουμε σε μια επιθυμητή κατάσταση με ενέργεια E, που περιγράφει σωστά το δίκτυο, κάνουμε μεταβολές στην παράμετρο T,μέσω των αλλαγών των βαρών. Αυτή η αρχική προσέγγιση των Μηχανών Boltzmann δεν είναι πρακτικά εφαρμόσιμη καθώς για να φτάσουμε στην σωστή ενέργεια απαιτείται πολύ μεγάλος χρόνος εκπαίδευσης. Ειδικότερα σε πολυεπίπεδα δίκτυα που χρησιμοποιούνται για προβλήματα όπου το δίκτυο πρέπει να μάθει να παράγει σωστά διανύσματα δεδομένων με μεγάλη πιθανότητα, χρειάζονται πολλές μικρές μεταβολές στα βάρη[12].

3.2 Restricted Boltzmann Machines

Το πρόβλημα εκπαίδευσης των BM έρχονται να λύσουν οι RBM. Οι RBM αποτελούνται από μόνο δύο επίπεδα νευρώνων και οι συνδέσεις μεταξύ αυτών είναι περιορισμένες (restricted). Κάθε νευρώνας ενός επιπέδου συνδέεται με όλους τους νευρώνες του απέναντι επιπέδου και μόνο αυτούς. Αυτό σημαίνει ότι δεν υπάρχουν συνδέσεις μεταξύ νευρώνων του ίδιου επιπέδου και άρα η κατάσταση κάθε νευρώνα εξαρτάται μόνο από το απέναντι επίπεδο. Με μια πρώτη ματιά καταλαβαίνουμε ότι ήδη το μέγεθος των υπολογισμών σε μια RBM είναι μειωμένο καθώς μειώνονται οι εξαρτήσεις μεταξύ του συνόλου των νευρώνων.

Η δύναμη των RBM κρύβεται στον ορισμό τους σύμφωνα με τον οποίο αποτελούν ενεργειακά στοχαστικά γενετικά μοντέλα. Δηλαδή η κατάστασή τους περιγράφεται από μια τιμή ενέργειας, κάθε νευρώνας παίρνει τιμή βασιζόμενος σε μια κατανομή πιθανότητας για μια συνάρτηση ενέργειας και τέλος είναι ικανά να παράγουν δεδομένα που να ακολουθούν αυτήν την πιθανολογική κατανομή.

Πριν προχωρήσουμε στην επεξήγηση του τρόπου εκπαίδευσής τους πρέπει να κατανοήσουμε βασικές έννοιες και την τοπολογία ενός RBM όπως δίνονται παρακάτω.



Εικόνα 8 Γραφική αναπαράσταση RBM

Μια Restricted Boltzmann Machine αποτελείται από ένα πλήθος ορατών (visible) μονάδων επεξεργασίας m, οι οποίες αποτελούν την είσοδο των δεδομένων μας, και πλήθος κρυφών (hidden) μονάδων n οι οποίες προσπαθούν να ανακαλύψουν τις συσχετίσεις - εξαρτήσεις μεταξύ των παρατηρούμενων στην είσοδο δεδομένων[10][13][14][15].

Όπως προείπαμε οι RBM αποτελούν παραλλαγή των BM και είναι και αυτές με την σειρά τους ενεργειακά μοντέλα (energy based models). Έτσι δεδομένης μιας παρατηρούμενης κατάστασης του δικτύου, η ενέργεια του συνδυασμού της διάταξης ορατών και κρυφών μονάδων δίνεται από την ακόλουθη συνάρτηση:

$$E(v,h) = -\sum_{i=1}^{m} a_i v_i - \sum_{j=1}^{n} b_j h_j - \sum_{i=1}^{m}\sum_{j=1}^{n} v_i w_{I,j} h_j$$

Όπου :

- v είναι ένα δυαδικό διάνυσμα πλήθους m της κατάστασης των ορατών μονάδων
- h ένα δυαδικό διάνυσμα πλήθους n της κατάστασης των κρυφών μονάδων
- Vi είναι η κατάσταση της ορατής μονάδας i
- Hj είναι η κατάσταση της κρυφής μονάδας j
- ai είναι η προδιάθεση (bias) της ορατής μονάδας i
- bj είναι η προδιάθεση της κρυφής μονάδας j

Ακολουθώντας τις αρχές που διέπουν την εκπαίδευση ενός RBM,όπως αυτές παρουσιάζονται στον οδηγό του Hinton [16], έχουμε ότι :

Οι κατανομές πιθανότητας που ορίζονται με βάση την συνάρτηση ενέργειας έχουν την μορφή $P(x) = \frac{e^{-Energy(x)}}{Z}$.

όπου Z καλείται partition function και είναι στην ουσία ένας παράγοντας κανονικοποίησης με $Z = \sum_x e^{-Energy(x)}$.

Η πιθανότητα που το δίκτυο αναθέτει σε κάθε δυνατό ζευγάρι ορατών - κρυφών διανυσμάτων δίνεται από την σχέση :

$p(v,h) = \frac{e^{-E(v,h)}}{Z}$,όπου το Z δίνεται από το άθροισμα όλων των ζευγών των διανυσμάτων των ορατών και κρυφών νευρώνων, δηλαδή $Z = \sum_{v,h} e^{-E(v,h)}$.

Στην περίπτωση που εστιάζουμε σε ένα διάνυσμα του ορατού επιπέδου, η πιθανότητα που του αποδίδεται από το δίκτυο αντιστοιχεί στο άθροισμα όλων των κρυφών διανυσμάτων :

$p(v) = \frac{1}{Z} \sum_h e^{-E(v,h)}$.

Με βάση ένα σύνολο δεδομένων προς εκπαίδευση (training cases), η λειτουργία που πραγματοποιεί το RBM είναι η εύρεση τιμών για εκείνες τις παραμέτρους του, που μεγιστοποιούν την λογαριθμική πιθανότητα εμφάνισης του συγκεκριμένου συνόλου. Οι παράμετροι αυτοί είναι οι τιμές των συναπτικών βαρών και των προδιαθέσεων των μονάδων.

Η παράγωγος της λογαριθμικής πιθανότητας ενός διανύσματος συναρτήσει ενός βάρους (w),στην περίπτωση αυτή, είναι απλή και δίνεται από την σχέση :

$\frac{\vartheta log p(v)}{\vartheta w_{ij}} = \langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}$ όπου τα <> συμβολίζουν τις προσδοκίες της αντίστοιχης κατανομής δεδομένων. Αυτό μας οδηγεί σε έναν πολύ απλό κανόνα εκπαίδευσης εκτελώντας απότομη στοχαστική ανύψωση (steepest ascent) στην λογαριθμική πιθανότητα των δεδομένων εκπαίδευσης (training data) με : $\Delta \boldsymbol{w_{ij}} = \epsilon\left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{model}\right)$ όπου ϵ είναι ο ρυθμός εκπαίδευσης (learning rate).

Επειδή δεν υπάρχουν άμεσες συνδέσεις μεταξύ των κρυφών μονάδων μπορούμε να πάρουμε ένα μη-προδιαθετημένο (unbiased) δείγμα του $\langle v_i j_j \rangle_{data}$ σχετικά εύκολα,αφού ξέροντας τις τιμές των μονάδων στο ορατό επίπεδο μπορούμε να υπολογίσουμε την δεσμευμένη πιθανότητα για κάθε μονάδα να είναι 1 στο κρυμμένο επίπεδο $p\left(h_j = 1 \mid v\right) = \sigma\left(b_j + \sum_i v_i w_{ij}\right)$. Και επίσης επειδή δεν υπάρχουν άμεσες συνδέσεις μεταξύ των ορατών μονάδων, μπορούμε να πάρουμε ένα μη-προδιαθετημένο δείγμα της κατάστασης μια ορατής μονάδας $p(v_i = 1 \mid h) = \sigma\left(a_j + \sum_j h_j w_{ij}\right)$,δεδομένου ενός κρυφού διανύσματος, όπου σ η σιγμοειδής συνάρτηση 1/(1 + exp(-x)). Τότε το $v_i h_j$ αποτελεί ένα δείγμα χωρίς προδιάθεση .

Το να αποκτήσουμε ένα μη-προδιαθετημένο δείγμα του μοντέλου, $\langle v_i h_j \rangle_{model}$, είναι πολύ πιο δύσκολο. Παρ' όλα αυτά είναι εφικτό, με το να εφαρμόσουμε δειγματοληψία Gibbs (Gibbs sampling) στις μονάδες του ορατού επιπέδου αφού τις αρχικοποιήσουμε σε μια τυχαία κατάσταση. Μια εναλλασσόμενη επανάληψη δειγματοληψίας Gibbs όμως συνιστά την ενημέρωση παράλληλα όλων των κρυφών μονάδων και έπειτα όλων των ορατών, χρησιμοποιώντας τις δύο προηγούμενες συναρτήσεις. Επιπλέον η διαδικασία θα πρέπει να επαναληφθεί για μεγάλο χρονικό διάστημα (τείνει στο άπειρο), ώστε να αναγκαστεί το μοντέλο να "ξεχάσει" την τυχαία κατάσταση στην οποία το αρχικοποιήσαμε.

Η εκπαίδευση ενός RBM βασίζεται κατά αντιστοιχία στον τρόπο που εκπαιδεύονται τα ενεργειακά μοντέλα. Έτσι θεωρώντας ότι υπάρχει μια συγκεκριμένη τιμή ενέργειας που περιγράφει με ακρίβεια το μοντέλο μας, τότε αυτή καθορίζεται από τις τιμές των παραμέτρων αυτού (συναπτικά βάρη). Θέλοντας να προσαρμόσουμε όσο το δυνατόν καλύτερα αυτές τις παραμέτρους τις μεταβάλλουμε και με βάση ενός συνόλου παραδειγμάτων – δεδομένων (training set) προσπαθούμε να μεγιστοποιήσουμε την πιθανότητα εμφάνισης αυτών, δίνοντάς τους χαμηλότερη ενέργεια. Για να καταλάβουμε λίγο καλύτερα την έννοια της ενέργειας στα RBM θα μπορούσαμε να πούμε ότι εκπαιδεύουμε μια RBM ώστε να μας δίνει ως έξοδο μια χαμηλή τιμή ενέργειας για μια διάταξη ορατών – κρυφών νευρώνων που εμφανίζεται συχνά στο σύνολο των δεδομένων εκπαίδευσης (training data) μας.

3.2.1 Contrastive divergence

Το να εφαρμόσουμε δειγματοληψία Gibbs με πολλές επαναλήψεις, μέχρι το σύστημα να έρθει σε ισορροπία, είναι κάτι πρακτικά ανέφικτο καθώς απαιτείται πολύς χρόνος. Το γεγονός αυτό

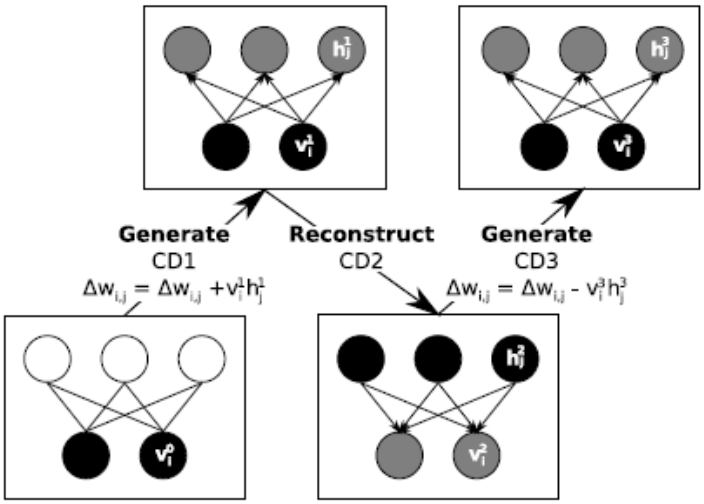
είχε κρατήσει στον πάγο τα RBMs για αρκετό χρονικό διάστημα. Το 2002 ο Hinton βρήκε έναν άλλον τρόπο να συλλέξει τα στατιστικά από το μοντέλο με μια μέθοδο που ονομάζει Contrastive Divergence[10][16]. Η ιδέα πίσω από την Contrastive Divergence μέθοδο είναι απλή και λειτουργεί πολύ καλά στην πράξη. Αντί να αρχικοποιήσουμε το ορατό επίπεδο σε τυχαίες τιμές, το αρχικοποιούμε με τιμές από ένα παράδειγμα προς εκπαίδευση (training example). Έπειτα υπολογίζουμε την δυαδική κατάσταση όλων των κρυφών μονάδων παράλληλα χρησιμοποιώντας την προηγούμενη σχέση $p(h_j = 1 \mid v) = \sigma\left(b_j + \sum_i v_i w_{ij}\right)$. Εφόσον έχουν υπολογιστεί οι δυαδικές καταστάσεις των κρυφών μονάδων ξεκινάει μια φάση «ανοικοδόμησης» (reconstruction) όπου θέτουμε κάθε μονάδα Vi του ορατού επιπέδου σε 1 με πιθανότητα που δίνεται από την σχέση $p(v_i = 1 \mid h) = \sigma\left(a_j + \sum_j h_j w_{ij}\right)$. Η αλλαγή στα βάρη,τότε, δίνεται από την σχέση $\Delta w_{ij} = \in \left(\langle v_i h_j \rangle_{data} - \langle v_i h_j \rangle_{recon}\right)$. Στην πραγματικότητα αυτή η μέθοδος είναι μια «πρόχειρη» προσέγγιση της κλίσης της λογαριθμικής πιθανότητας,και είναι στην ουσία μια δειγματοληψία Gibbs που τρέχει για πολύ μικρό αριθμό επαναλήψεων. Παρ όλα αυτά στην πράξη δουλεύει πολύ καλά και συνήθως χρησιμοποιείται μια μόνο επανάληψη (Generate – Reconstruct – Generate). Στις παρακάτω εικόνες παρουσιάζεται μια γενική μορφή της CD μεθόδου σε αλγοριθμικό και σχηματικό επίπεδο.

Algorithm 1. k-step contrastive divergence

Input: RBM $(V_1, \ldots, V_m, H_1, \ldots, H_n)$, training batch S
Output: gradient approximation Δw_{ij}, Δb_j and Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

1 init $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
2 forall the $v \in S$ do
3 $v^{(0)} \leftarrow v$
4 for $t = 0, \ldots, k-1$ do
5 for $i = 1, \ldots, n$ do sample $h_i^{(t)} \sim p(h_i \mid v^{(t)})$
6 for $j = 1, \ldots, m$ do sample $v_j^{(t+1)} \sim p(v_j \mid h^{(t)})$
7 for $i = 1, \ldots, n$, $j = 1, \ldots, m$ do
8 $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 \mid v^{(0)}) \cdot v_j^{(0)} - p(H_i = 1 \mid v^{(k)}) \cdot v_j^{(k)}$
9 $\Delta b_j \leftarrow \Delta b_j + v_j^{(0)} - v_j^{(k)}$
10 $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 \mid v^{(0)}) - p(H_i = 1 \mid v^{(k)})$

Εικόνα 9 Αλγόριθμος Contrastive Divergence για κ – βήματα [10]



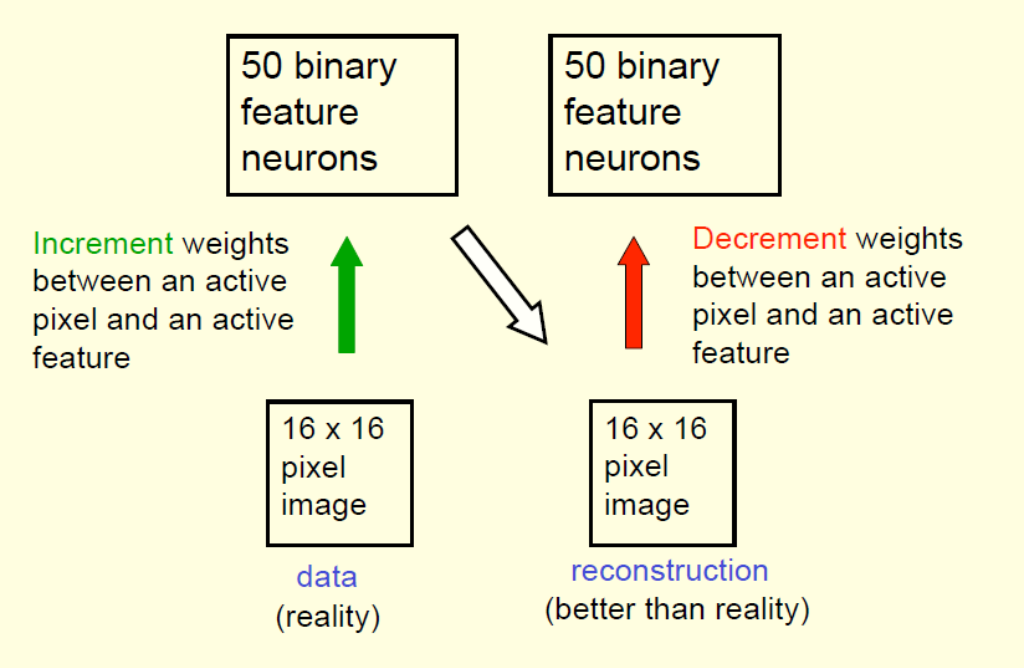
Εικόνα 10 Σχηματική απεικόνιση Contrastive Divergence [26]

Όπως γίνεται κατανοητό τα RBM πλέον εκπαιδεύονται με αυτήν την μέθοδο κάτι που έχει δώσει νέα ώθηση στον κλάδο της Μηχανικής Μάθησης, καθώς πλέον είναι εφικτή η εκπαίδευσή τους σε ικανοποιητικούς χρόνους.

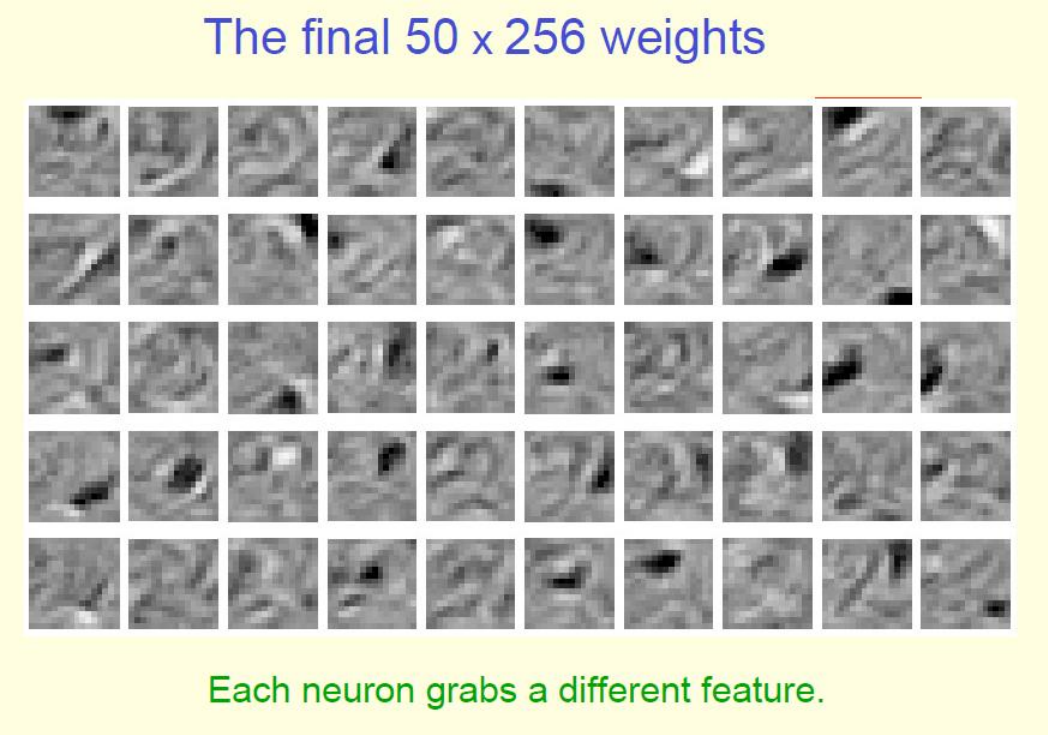
3.3 Deep Belief Networks

Τα RBMs μπορούν να χρησιμοποιηθούν με διάφορους τρόπους. Μπορούν να χρησιμοποιηθούν για ταξινόμηση, ανακατασκευή, για πρόβλεψη και εξαγωγή χαρακτηριστικών. Η πιο σημαντική χρήση που βρίσκουν είναι ως ανιχνευτές χαρακτηριστικών για Δίκτυα Βαθιάς Πεποιθήσεως (Deep Belief Networks). Ένα DBN είναι μια «στοίβα» απλούστερων νευρωνικών δικτύων, συνδεδεμένα μεταξύ τους. Τα DBNs αποτελούν και αυτά με την σειρά τους πιθανολογικά γενετικά μοντέλα καθώς αποτελούνται από πολλών επιπέδων στοχαστικές μεταβλητές οι οποίες συχνά αναφέρονται και ως ανιχνευτές χαρακτηριστικών (feature detectors)[18],[21].Αυτό γιατί ο κορμός τους είναι είτε RBMs είτε autoencoders. Περιέχουν πολλά επίπεδα RBM όπου τα χαμηλότερα επίπεδα έχουν κατευθυνόμενες συνδέσεις προς αυτά από τα πιο πάνω επίπεδα, ενώ τα δύο υψηλότερα επίπεδα έχουν συμμετρικές μη κατευθυνόμενες συνδέσεις μεταξύ τους σχηματίζοντας μια συσχετιστική μνήμη. Στο χαμηλότερο επίπεδο εφαρμόζουμε το διάνυσμα δεδομένων εισόδου. Η λειτουργία της εκπαίδευσης ενός DBN [22],[23] γίνεται ανά επίπεδο ξεκινώντας από πρώτο επίπεδο το οποίο εκπαιδεύεται ως ένα RBM που μοντελοποιεί τα δεδομένα που δέχεται σαν είσοδο στο ορατό του επίπεδο. Έχοντας μοντελοποιήσει τα δεδομένα χρησιμοποιεί τις αντίστοιχες τιμές στο κρυφό του επίπεδο ως είσοδο για το ορατό επίπεδο του επόμενου RBM. Ανάλογα με το πόσα επίπεδα έχουμε αυτή η διαδικασία επαναλαμβάνεται διαδίδοντας στα επόμενα επίπεδα, μοντέλα επί μοντέλων των αρχικών επιπέδων. Στο τελευταίο επίπεδο θα γίνει μια βελτιστοποίηση, συνήθως με οπισθοδιάδοση (backpropagation), όλων των παραμέτρων (βαρών) για την καλύτερη απόδοση του δικτύου.

Ακολουθεί ένα παράδειγμα του τρόπου που δομείται και λειτουργεί ένα DBN. Στο παράδειγμα αυτό,όπως φαίνεται στο [24], θα χρησιμοποιήσουμε DBN για την αναγνώριση ψηφίων, γραμμένων με το χέρι, των αριθμών από το 0 ως το 9. Είπαμε πως η βάση των DBN είναι τα RBM,οπότε σε μια τα τέτοια περίπτωση ένα RBM θα εξάγει χαρακτηριστικά για ένα διάνυσμα από pixels ενός ψηφίου με την μέθοδο Contrastive Divergence (generate – reconstruct – generate) οπότε στα βάρη του θα έχουν αποθηκευθεί οι σωστές τιμές που αναπαριστούν το συγκεκριμένο ψηφίο και θα έχει κρατήσει στους νευρώνες του διαφορετικά χαρακτηριστικά που αφορούν στο συγκεκριμένο ψηφίο .Όπως φαίνεται παρακάτω για ένα διάνυσμα 16 x 16 pixels του ψηφίου 2 :



Εικόνα 11RBM για είσοδο 16x16 pixels [24]



Εικόνα 12 Χαρακτηριστικά που αποθηκεύονται στους νευρώνες [24]

Βλέπουμε ότι οι 50 κρυφοί νευρώνες έχουν εξάγει διαφορετικά χαρακτηριστικά που αφορούν στον σχηματισμό του ψηφίου 2.

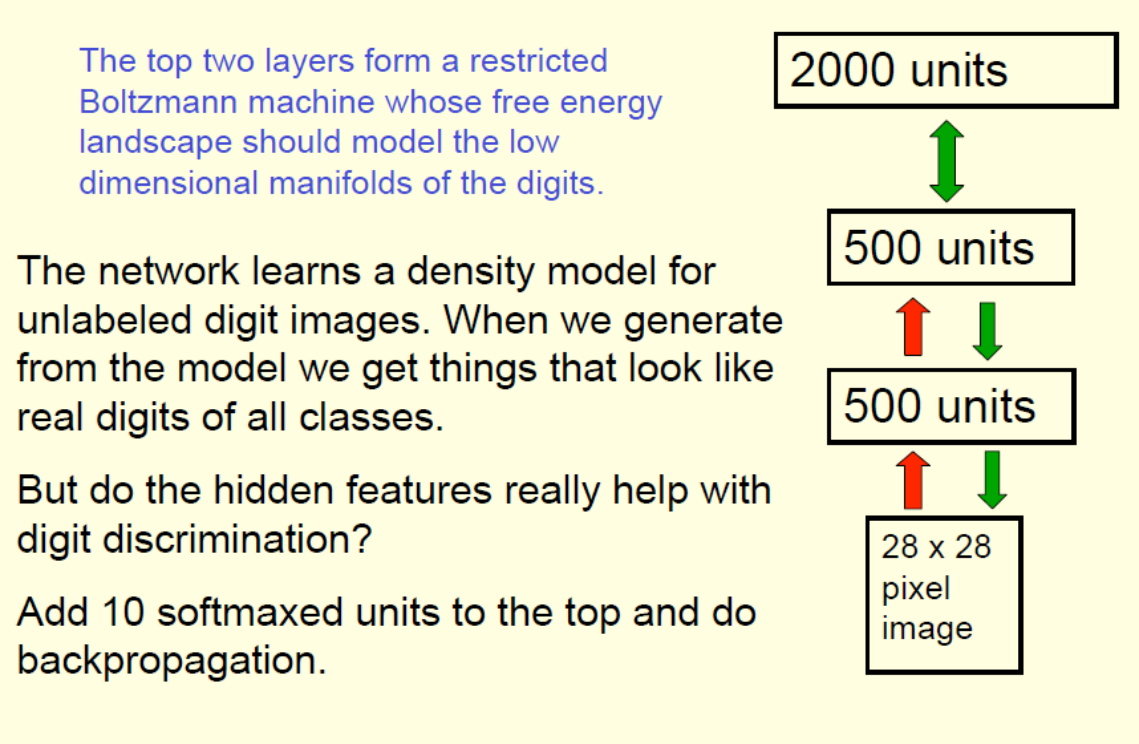
Για κάθε ψηφίο, το RBM θα εκπαιδευτεί με τον ίδιο τρόπο και θα δώσει ίδιου τύπου αποτελέσματα. Σε ένα DBN τα δεδομένα των κρυφών νευρώνων αποτελούν είσοδο σε επόμενο

RBM και αυτό με την σειρά του θα εξάγει χαρακτηριστικά για αυτό το διάνυσμα. Το τελικό μοντέλο μας για τα 10 ψηφία θα μπορεί «να γεννάει» δεδομένα που μοιάζουν με αληθινά ψηφία από όλες τις κλάσεις. Αυτή η λειτουργία ενός DBN μπορεί να βελτιστοποιηθεί χρησιμοποιώντας μία εκδοχή του αλγόριθμου wake-sleep όπου :

1.Προσαρμόζουμε τα βάρη (που διαδίδονται προς τα κάτω) στα κάτω επίπεδα για καλύτερη εξαγωγή χαρακτηριστικών

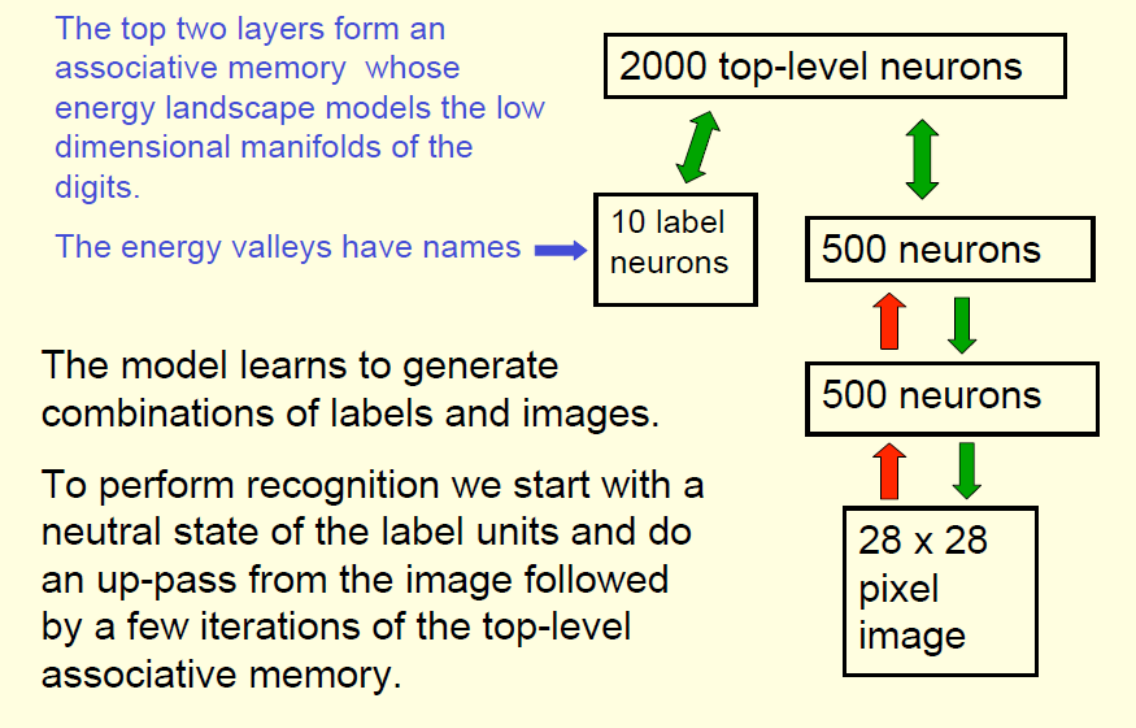
2.Κάνουμε μερικές επαναλήψεις για να πάρουμε δείγματα στο RBM του υψηλότερου επιπέδου, αλλάζοντας και εκεί τα βάρη

3.Κάνουμε ένα πέρασμα από τα πάνω προς τα κάτω επίπεδα ώστε να βελτιστοποιήσουμε τα βάρη που διαδίδονται προς τα πάνω και έτσι παράγουν (reconstruct) καλύτερα αποτελέσματα για πιο πάνω επίπεδα.



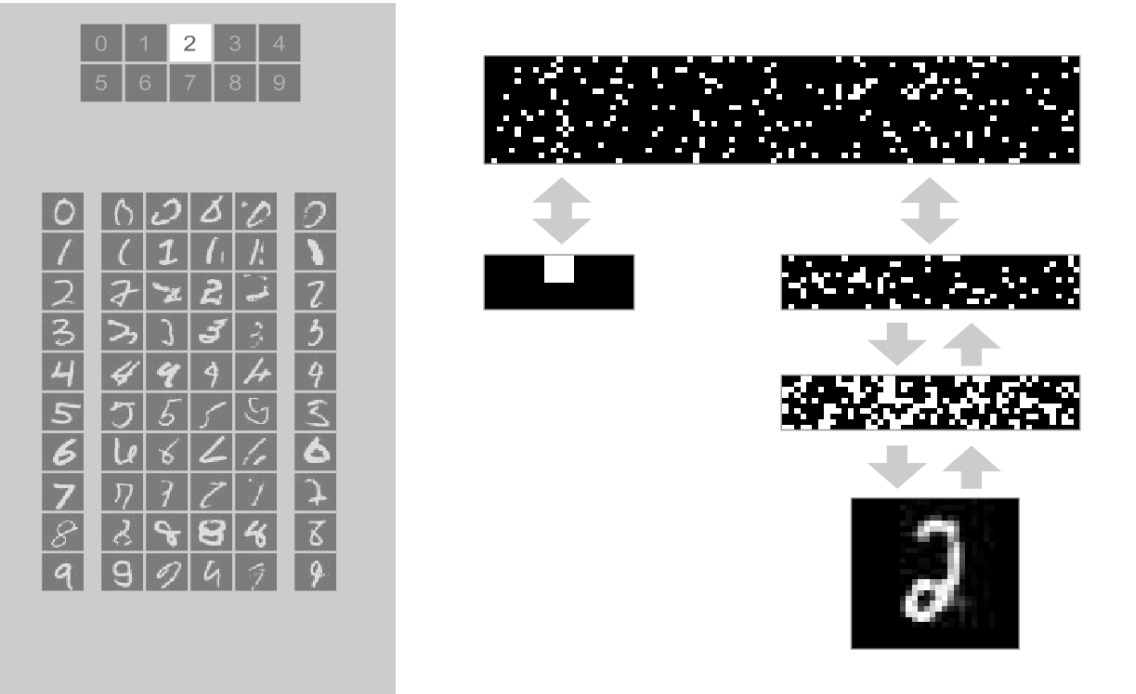
Εικόνα 13 DBN για γένεση ψηφίων από 10 κλάσεις [24]

Αυτή είναι η «γενετική» λειτουργία ενός DBN. Για ταξινόμηση πραγματοποιούμε μια παραλλαγή στο μοντέλο μας. Προσθέτουμε ένα επιπλέον επίπεδο με 10 νευρώνες (που αναπαριστούν τις 10 κλάσεις ψηφίων που έχουμε) και αλλάζουμε τον αλγόριθμο βελτιστοποίησης των βαρών σε οπισθοδιάδοση (backpropagation) . Έτσι πλέον έχουμε ένα DBN ικανό για αναγνώριση / ταξινόμηση ψηφίων γραμμένων με το χέρι.



Εικόνα 14 DBN για αναγνώριση ψηφίων από τις 10 κλάσεις [24]

Μια εικονική αναπαράσταση της διαδικασίας μπορεί να βρεθεί στο [25].Για το ψηφίο 2 μπορούμε να δούμε παρακάτω πως συσχετίζεται η είσοδος στο κάτω επίπεδο με την ταξινόμηση του στην κλάση 2 στο τελευταίο επίπεδο. Τα ενδιάμεσα επίπεδα είναι RBM που εκτελούν την προαναφερθείσα διαδικασία εξαγωγής χαρακτηριστικών και ταξινόμησης.



Εικόνα 15 Εικονική αναπαράσταση λειτουργίας DBN για αναγνώριση του ψηφίου 2

4. RBMs και Hardware

Όπως καταλαβαίνουμε, τα DBN είναι πολύ χρήσιμα στην εξαγωγή γνώσης από αταξινόμητα δεδομένα. Παραλλαγές του DBN που δείξαμε προηγουμένως χρησιμοποιούνται σε μια μεγάλη γκάμα περιπτώσεων Μηχανικής Μάθησης και αποτελούν state of the art αλγόριθμο για εφαρμογές speech recognition, hand written digits recognition, documents classification, αναγνώρισης χαρακτηριστικών προσώπων και πολλών άλλων.

Παρ' όλα αυτά για να εκπαιδευτούν σωστά τα DBNs απαιτείται συνήθως μεγάλος χρόνος εκπαίδευσης και αρκετοί πόροι (μνήμη και επεξεργαστική ισχύς). Αυτό έχει οδηγήσει σε μια προσπάθεια να μεταφερθούν οι υπολογισμοί ενός RBM,και κατ' επέκταση ενός DBN, σε επίπεδο υλικού.

Γενικότερα έχουν γίνει προσπάθειες μεταφοράς Τεχνητών Νευρωνικών δικτύων σε hardware αλλά οι κλασσικές feed forward / backpropagation υλοποιήσεις τέτοιων δικτύων δεν είναι αποδοτικές σε σχεδιάσεις υλικού. Αυτό οφείλεται στο γεγονός ότι αυτές οι αρχιτεκτονικές χρησιμοποιούν αριθμητική πραγματικών αριθμών και απαιτούν «βαριά» συστατικά υλικού όπως πολλαπλασιαστές και συσσωρευτές. Επιπλέον έχουν πολύπλοκες συναρτήσεις μεταφοράς (transfer functions) και κάθε στοιχείο επεξεργασίας χρειάζεται αρκετούς πόρους υλικού για να επιτευχθεί η λειτουργία του. Καταλήγοντας έτσι στην σχεδίαση custom pipeline αρχιτεκτονικής, ανάλογα με την περίσταση, που δεν είναι ικανή να παρέχει παραλληλισμό ή χρόνους εκπαίδευσης τέτοιους που να δικαιολογούν το κόστος για την υλοποίηση αυτών [26].

Με την εδραίωση όμως των DBN ως κύριο τρόπο επίλυσης δύσκολων προβλημάτων Μηχανικής Μάθησης ,οι υλοποιήσεις ΤΝΔ σε hardware επίπεδο γίνονται ολοένα πιο αποδοτικές. Τα RBMs είναι καταλληλότερα για hardware υλοποιήσεις είτε αυτό πρόκειται για FPGAs είτε για GPUs.

Όσο αφορά στα FPGAs τα RBMs χρησιμοποιούν τύπους δεδομένων που αντιστοιχίζονται πολύ καλά σε δομές hardware καθώς οι καταστάσεις των κόμβων έχουν δυαδικές τιμές και οι λειτουργίες που εκτελούνται μπορούν να γίνουν με απλές πύλες αντί πολλαπλασιαστών. Επιπλέον δεν χρειάζονται μεγάλη ακρίβεια και μπορεί να χρησιμοποιηθεί αριθμητική σταθερής υποδιαστολής (fixed point arithmetic) αυξάνοντας τις επιδόσεις και μειώνοντας την χρήση πόρων [26],[27]. Τέλος η απλότητα της αρχιτεκτονικής τους μπορεί να οδηγήσει σε «έξυπνες» σχεδιάσεις υλικού και σε αντίθεση με ASIC σχεδιάσεις, μια σχεδίαση σε FPGA έχει το σημαντικό πλεονέκτημα ότι είναι παραμετροποιήσιμη ανάλογα με την τοπολογία του δικτύου που θέλουμε για την εκάστοτε εφαρμογή [31],[32].

Όσο αφορά στις GPUs δίνεται έμφαση στην μαζική παραλληλία που είναι έμφυτη στα νευρωνικά δίκτυα και ιδίως στις RBM μηχανές. Η παράλληλοι επεξεργαστές σε μια GPU λειτουργούν αποδοτικότερα από αυτές των γενικού σκοπού και υπάρχει δυνατότητα να τρέχουν παράλληλα περισσότερα νήματα. Επιπλέον, οι σύγχρονες κάρτες γραφικών έχουν πολύ δυνατούς επεξεργαστές και μεγάλη χωρητικότητα μνήμης, κάτι που τις κάνει αρκετά αποδοτικές στην εκτέλεση παράλληλων λειτουργιών και υπολογισμών, όπως ενός RBM. Τα προβλήματα που πρέπει να ξεπεραστούν στις GPU σχεδιάσεις των RBM έχουν να κάνουν κυρίως με τον τρόπο προσπέλασης της μνήμης ώστε να είναι αποδοτικές οι πράξεις μεταξύ πινάκων που χρησιμοποιούνται κατά κόρον [28],[29].

4.1 RBMs και FPGAs

Σε τεχνολογία FPGA γίνονται συνεχείς προσπάθειες για καλύτερες σχεδιάσεις που θα οδηγήσουν σε ακόμα καλύτερες υλοποιήσεις RBM.Οι αρχικές προσεγγίσεις συνοψίζουν τα υπέρ και τα κατά τέτοιων υλοποιήσεων και θέτουν τις βάσεις για επόμενες σχεδιάσεις. Δίνουν τα πρώτα ενθαρρυντικά αποτελέσματα και στις ακόλουθες παραγράφους θα προσπαθήσουμε να τις αποτιμήσουμε και εμείς με την σειρά μας, με σκοπό να διακρίνουμε που προσφέρουν και που υστερούν ώστε να μπορέσουμε να εστιάσουμε σε μια διαφορετική προσέγγιση.

4.1.1 Έμφαση στην ταχύτητα

Στα [26],[27],[30] οι συγγραφείς δεν προσπαθούν να φτιάξουν ακριβώς μια RBM αλλά εντάσσουν τις λειτουργίες του σε ένα πλαίσιο hardware με έμφαση στην ταχύτητα και την μετατροπή του προβλήματος από O(n^2) σε O(n). Η προσέγγιση αυτή βασίζεται στην ανάλυση των συναρτήσεων που απαρτίζουν το πρόβλημα και την επαναδιατύπωσή αυτών ώστε να βρεθεί το σημείο που χρειάζεται επιτάχυνση. Έτσι μεταφράζουν τις λειτουργίες μιας RBM σε πράξεις μεταξύ διανυσμάτων / πινάκων και προσπαθούν να βελτιστοποιήσουν αυτήν την συμπεριφορά. Επιπλέον, δίνεται έμφαση στην δυαδική φύση των κόμβων και χρησιμοποιούν απλές πύλες για αριθμητικές λειτουργίες μεταξύ αυτών. Η τοπικότητα των δεδομένων παίζει μεγάλο ρόλο,και η συγκεκριμένη σχεδίαση χρησιμοποιεί πολλά BRAMs για τοπική αποθήκευση. Στην ανάλυση που πραγματοποιείται γίνεται σύνοψη όλων των τύπων που χρησιμοποιούνται στο RBM όπως φαίνεται παρακάτω :

$E = -\sum_{i,j} w_{i,j}\, v_i\, h_j$ (1)

$E = \sum_i v_i\left(-\sum_j w_{i,j}\, h_j\right) = \sum_i v_i E_i$ (2)

$= \sum_j h_j\left(-\sum_i w_{i,j}\, v_i\right) = \sum_j h_j E_j$ (3)

$p(v_i = 1) = \frac{1}{1+ e^{E_i/T}}$ (4)

$p(h_j = 1) = \frac{1}{1+ e^{E_j/T}}$ (5)

$v_i = \begin{cases} 0\,, & E_i < 0 \\ 1\,, & E_i \geq 0 \end{cases}$ (6)

$h_j = \begin{cases} 0\,, & E_j < 0 \\ 1\,, & E_j \geq 0 \end{cases}$ (7)

$\frac{\partial E}{\partial w_{i,j}} = \frac{1}{T}\left(\langle v_i h_j\rangle^1 - \langle v_i h_j\rangle^\infty\right)$ (8)

$w_{i,j}[k+1] = w_{i,j}[k] + \epsilon\left(\langle v_i h_j\rangle^1 - \langle v_i h_j\rangle^k\right)$ (9)

$\langle v_i h_j\rangle^X = \frac{1}{l}\sum_{l=0}^{l} {v_i}^X {h_j}^X$ (10)

Έπειτα γίνεται επαναδιατύπωσή τους σε μορφή πινάκων όπου για το ορατό και κρυφό επίπεδο έχουμε :

$v_l^X = [v_0^X \dots v_{i-1}^X] \in B^{1\times i}$, $h_l^X = [h_0^X \dots h_{j-1}^X] \in B^{1\times j}$

Και για ένα ολόκληρο batch τα επίπεδα μπορούν να γραφούν και ως :

$$V^X \begin{bmatrix} v_0^x \\ . \\ . \\ . \\ v_{l-1}^x \end{bmatrix} \in B^{1\times i}\,,\quad H^X \begin{bmatrix} h_0^x \\ . \\ . \\ . \\ h_{j-1}^x \end{bmatrix} \in B^{1\times j}$$

Τα βάρη αναπαρίστανται ως :

$$W[k] = \begin{bmatrix} w_{0,0}[k] & \cdots & w_{0,j}[k] \\ \vdots & \ddots & \vdots \\ w_{i,0}[k] & \cdots & w_{i,j}[k] \end{bmatrix} \in R^{i\times j}$$

Έτσι πλέον οι αρχικές εξισώσεις (1) – (10) μεταφράζονται αντίστοιχα στις παρακάτω :

$V^{X+1} = \begin{cases} V^0\,, X = 0 \\ f(E_v^X)\,, X\ is\ odd \\ V^X\,, X = is\ even \end{cases}$ (11)

$H^{X+1} = \begin{cases} f(E_h^X)\ , X\ is\ even \\ V^X\,, X = is\ odd \end{cases}$ (12)

$E_v^X = (H^X)W^T\,, \in R^{l\times i}$ (13)

$E_h^X = (V^X)W\,, \in R^{l\times j}$ (14)

$W[k+1] = W[k] + \frac{\epsilon}{l}((V^1)^T H^1 + (V^X)^T (H)^T)$ (15)

Ο αλγόριθμος του RBM που σχηματίζεται σε αυτήν την περίπτωση είναι ο παρακάτω :

```
for every batch :
    visible[] = get_datavector(batch)

    for every AGS_phase :
        if AGS_phase is odd :
            # Energy compute Eq.14 - 2 loops -> O(n^2)
            for every hidden_node :
                for every visible_node :
                    energy[j] += visible[i]*weight[i][j]

            # Node select Eq.12 - 1 loop -> O(n)
            for every hidden_node :
                hidden[j] = transfer_function(energy[j])

        else :
            # Energy compute Eq.13 - 2 loops -> O(n^2)
            for every visible_node :
                for every hidden_node :
                    energy[i] += hidden[j]*weight[i][j]

            # Node select Eq.11 - 1 loop -> O(n)
            for every visible_node :
                visible[i] = transfer_function(energy[i])

        # Weight update Eq.15 - 2 loops -> O(n^2)
        if ABS_phase == 1 :
            for every visible_node :
                for every hidden_node :
                    weight_update[i][j] += visible[i] * hidden[j]
        else if ABS_phase == ABS_limit :
            for every visible_node :
                for every hidden_node :
                    weight_update[i][j] -= visible[i] * hidden[j]

# Weight update Eq.15 - 2 loops -> O(n^2)
for every visible_node :
    for every hidden_node :
        weight[i][j] += learning_rate/batch * weight_update[i][j]
```

Εικόνα 16 Αλγόριθμος RBM [26]

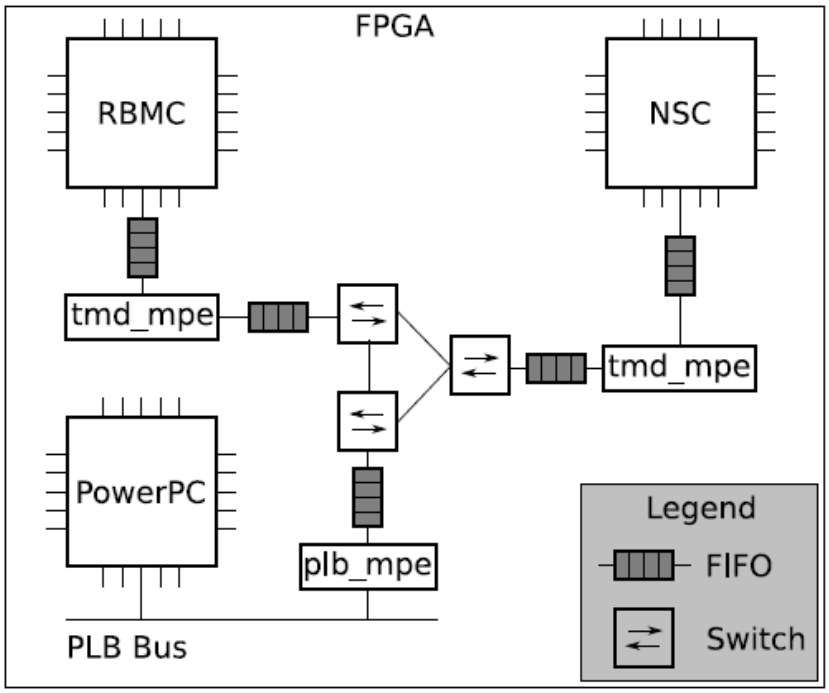
Με κόκκινο έχουμε σημειώσει τις πράξεις εκείνες που έχουν πολυπλοκότητα O(n^2) και τις οποίες οι συγγραφείς μετατρέπουν σε O(n) με χρήση «έξυπνων» και αποδοτικών δομών hardware.

Πριν προβούμε στην περαιτέρω ανάλυση της μεθόδου που χρησιμοποιείται πρέπει να ορίσουμε κάποιες παραδοχές / περιορισμούς που αφορούν στην συγκεκριμένη σχεδίαση :

1. Τα βάρη αποθηκεύονται σε αναπαράσταση σταθερού υποδιαστολής 32 bit
2. Τα επίπεδα είναι συμμετρικά, καθώς έτσι το ίδιο hardware χρησιμοποιείται για τον υπολογισμό των μερικών ενεργειών (13-14)
3. Ο αριθμός των κόμβων είναι δύναμη του 2 για αποδοτικότερη χρήση εξειδικευμένου hardware όπως binary trees και καλύτερη επιτάχυνση του αλγορίθμου
4. Ο αριθμός των batches είναι δύναμη του 2, γεγονός που επιτρέπει την διαίρεση στην εξίσωση (10) με χρήση αριθμητικής ολίσθησης ενός bit
5. Οι τιμές CD και του ρυθμού εκπαίδευσης δίνονται ως είσοδοι software

Με βάση τις προηγούμενες εξισώσεις, τον αλγόριθμο που θα επιταχυνθεί και τους περιορισμούς που προαναφέραμε, οι συγγραφείς [26], [27] σχεδίασαν κύριους πυρήνες υπολογισμών που επικοινωνούν μεταξύ τους μέσω ενός PowerPC πυρήνα και χρήση TMD –

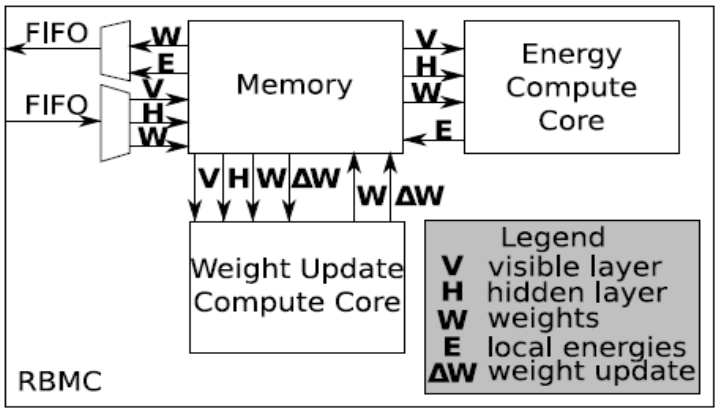
MPI πρωτοκόλλου. Το συγκεκριμένο πρωτόκολλο βασίζεται στο κλασικό Message Passing Interface που χρησιμοποιείται για την επικοινωνία επεξεργαστών σε παράλληλους υπολογισμούς, μέσω software. Το TMD αποτελεί μια παραλλαγή του που χρησιμοποιείται για την αντίστοιχη επικοινωνία μεταξύ επεξεργαστών σε επίπεδο hardware. Η γενική απεικόνιση της αρχιτεκτονικής φαίνεται παρακάτω :



Εικόνα 17 Απεικόνιση αρχιτεκτονικής [26]

Ο πυρήνας RBMC είναι το κυριότερο υπολογιστικό κομμάτι, υπεύθυνος για τον υπολογισμό των μερικών ενεργειών και την ανανέωση των βαρών. Ο πυρήνας NSC είναι υπεύθυνος για τον υπολογισμό της κατάστασης των κόμβων, με βάση τις μερικές ενέργειες. Ο PowerPC είναι υπεύθυνος για την εισαγωγή των δεδομένων και την προώθησή τους στους πυρήνες. Και η επικοινωνία όπως προείπαμε, γίνεται μέσω MPI και συγκεκριμένα TMD-MPI.

Ο Restricted Boltzmann Machine Core (RBMC) αποτελείται από τρείς επιμέρους δομές, όπως φαίνεται στην ακόλουθη εικόνα :



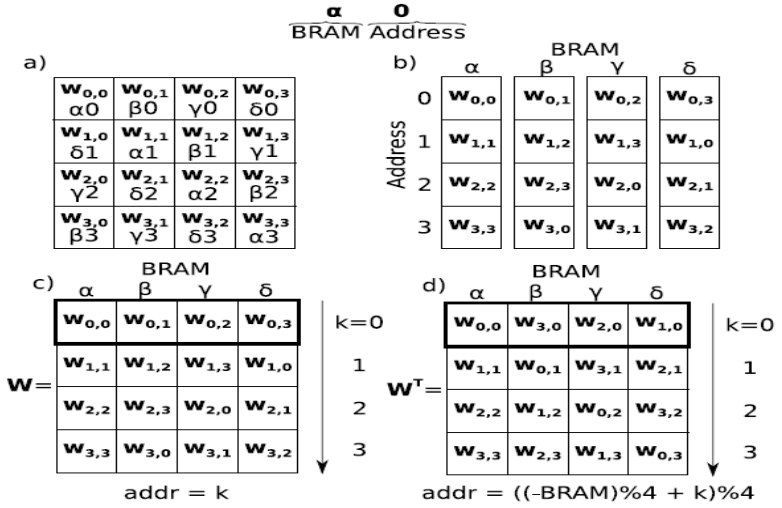
Εικόνα 18 Restricted Boltzmann Machine Core [26]

Ο πυρήνας Memory Core είναι στην ουσία το βασικότερο κομμάτι καθώς αποτελείται από BRAMs πάνω στο FPGA και επιτρέπει την παράλληλη εξαγωγή δεδομένων για άμεση χρήση από τους πυρήνες επεξεργασίας.

Είναι οργανωμένος ώστε να αποθηκεύει πέντε μεταβλητές έχοντας :

- 2 σετ καταχωρητών για το ορατό και κρυφό επίπεδο που παρέχουν παράλληλη και άμεση πρόσβαση σε όλους τους κόμβους
- 2 σετ από BRAMs για αποθήκευση των βαρών και την αναβάθμιση αυτών,και είναι dual ported επιτρέποντας ταυτόχρονα διάβασμα και γράψιμο (όσο δεν υπάρχουν conflicts)
- 1 BRAM για αποθήκευση μερικών ενεργειών

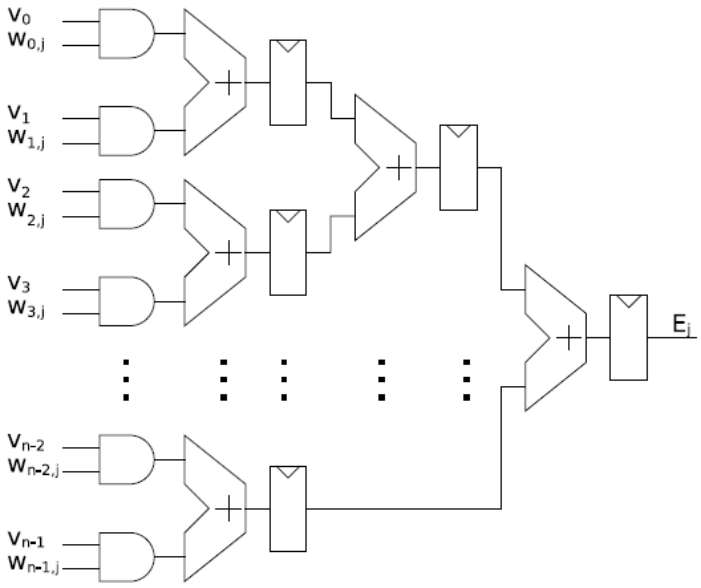
Τα BRAM των βαρών έχουν μια ιδιαίτερη δομή που επιτρέπει καλύτερη απόδοση, δεδομένου ότι οι προσβάσεις στην μνήμη δεν θα είναι τυχαίες. Επιπλέον έτσι αποφεύγουμε την δημιουργία του ανεστραμμένου πίνακα βαρών, εξίσωση (13) και πετυχαίνουμε πολλαπλασιασμό διανυσμάτων (rows ή columns) αντί για πολλαπλασιασμό πινάκων. Η δομή αυτή έχει να κάνει με τον τρόπο προσπέλασης της μνήμης και συνοψίζεται στην παρακάτω εικόνα :



Εικόνα 19 Τρόπος προσπέλασης της μνήμης [26]

Όπως παρατηρούμε οι εξισώσεις που έχουν O(n^2) πολυπλοκότητα είναι οι 13,14 και 15 οι οποίες έχουν να κάνουν με πολλαπλασιασμούς που περιέχουν τον πίνακα βαρών W είτε τον W ανεστραμμένο. Αποθηκεύοντας τον πίνακα W σε BRAMs με τρόπο τέτοιο ώστε από κάθε BRAM να μπορούμε να πάρουμε είτε μια ολόκληρη στήλη είτε μια ολόκληρη σειρά του W επιτυγχάνεται και η μετάβαση από πολλαπλασιασμό πινάκων σε πολλαπλασιασμό διανυσμάτων O(n) και η παράλληλη πρόσβαση που απαιτείται για τις αντίστοιχες λειτουργίες. Αυτό όμως σημαίνει την αφιέρωση n BRAMs πάνω στο FPGA,όπου n ο αριθμός των κόμβων.

Ο Energy Compute Core είναι υπεύθυνος για των υπολογισμό των ενεργειών (13-14) . Πολλαπλασιάζει το διάνυσμα ενός επιπέδου με μια σειρά ή στήλη του πίνακα W δημιουργώντας ένα στοιχείο στην στήλη του πίνακα ενεργειών. Λόγω των παραδοχών που κάναμε προηγουμένως ο υπολογισμός γίνεται με χρήση πυλών AND, multiplexers και registered fixed point adders ώστε να επιτευχθεί O(n) πολυπλοκότητα. Αυτό φαίνεται στην παρακάτω εικόνα :



Εικόνα 20 Energy Compute Core [26]

Ο Weight Update Compute Core είναι υπεύθυνος για τη επιτήρηση του όρου ανανέωσης του βάρους, και του υπολογισμού του. Κατά την πρώτη και τελευταία Alternate Gibbs Sampling φάση, διαβάζει παράλληλα τους καταχωρητές των επιπέδων και συσσωρεύει τον ρυθμό εκπαίδευσης (ΔW). Όταν ολοκληρωθεί το batch η συσσώρευση της ανανέωσης των βαρών δίνεται πίσω στον πίνακα βαρών (W). Πάλι χρησιμοποιούνται AND gates, multiplexers και fixed point αριθμητική και επειδή η μνήμη ανανεώνεται παράλληλα έχουμε χρόνο O(n).

Ο δεύτερος κύριος πυρήνας είναι ο Node Select Core,υπεύθυνος για τον υπολογισμό της κατάστασης των κόμβων βάση των μερικών ενεργειών (11-12).Δέχεται από τον RBMC τις τιμές των ενεργειών και επιστρέφει την κατάσταση των κόμβων. Για την υλοποίηση του προτείνονται δύο τρόποι.

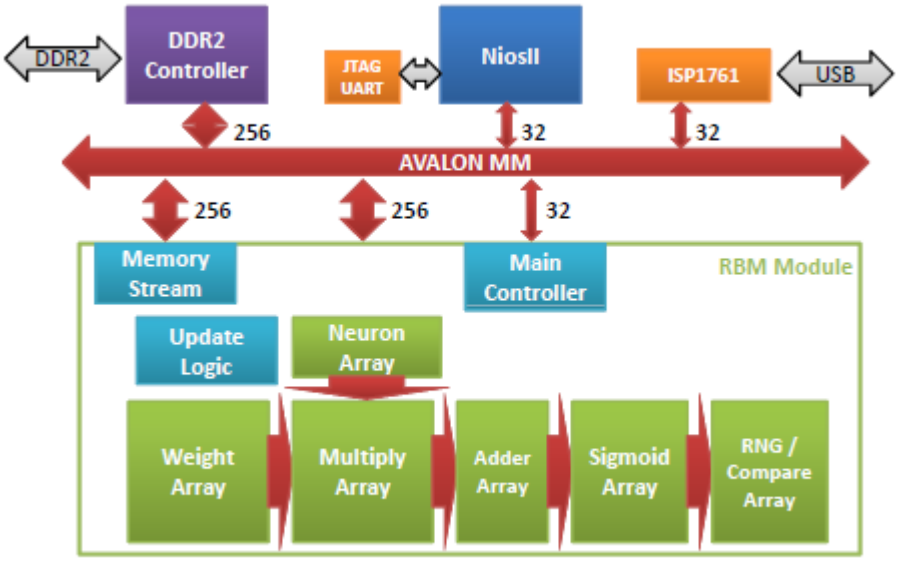
- Ένας ντετερμινιστικός απλός προσεγγιστικός τρόπος με μια συνάρτηση κατωφλίου (threshold function)
- Χρήση ενός πιο σωστού τρόπου όπου πετυχαίνει την στοχαστική επιλογή των κόμβων. Αυτό επιτυγχάνεται με την χρήση της τεχνικής Piecewise Linear Interpolation και την χρήση ενός BRAM ως LUT για την δημιουργία της σιγμοειδούς συνάρτησης. Έπειτα για την στοχαστική ανανέωση των κόμβων χρησιμοποιείται ο Tausworth -88 γεννήτορας τυχαίων αριθμών για σύγκριση και απόφαση ενεργοποίησης.

Σε επόμενες δημοσιεύσεις οι συγγραφείς κάνουν προσπάθειες ώστε να βελτιώσουν παραμέτρους της σχεδίασης που αφορούν τον αριθμό των κόμβων, την επεκτασιμότητα του μοντέλου και την επίτευξη καλύτερης προσέγγισης της σιγμοειδούς συνάρτησης.

4.1.2 Έμφαση στην επεκτασιμότητα

Στα [31],[32] συμπεραίνουν και εκεί οι συγγραφείς ότι στις λειτουργίες μιας RBM τον μεγαλύτερο ρόλο έχουν οι πολλαπλασιασμοί πινάκων. Όμως προχωρούν στην σχεδίασή τους με την λογική ότι στα σύγχρονα FPGA υπάρχει πλήθος διαθέσιμων πολλαπλασιαστών που μπορούν να χρησιμοποιηθούν για την επίτευξη των υπολογισμών μεταξύ των πινάκων . Επιπλέον η σχεδίασή τους γίνεται με γνώμονα την επέκτασή της σε περισσότερα FPGA και η δομή της προσπαθεί να είναι όσο το δυνατόν πιο τμηματική (modular).

Η γενική αρχιτεκτονική φαίνεται στην παρακάτω εικόνα :

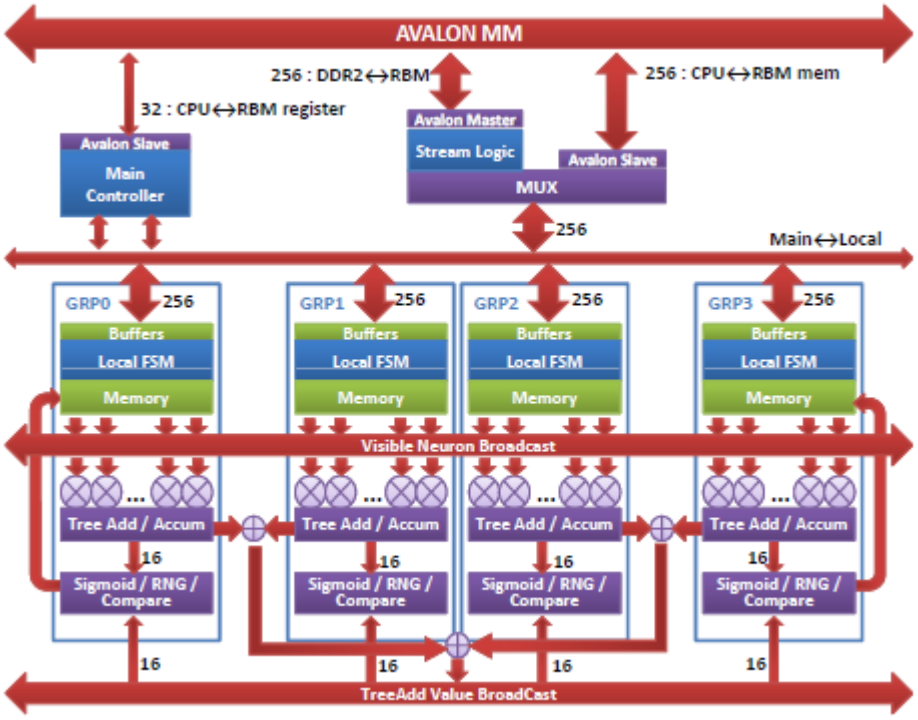


Εικόνα 21 Απεικόνιση αρχιτεκτονικής [31]

Η σχεδίαση περιλαμβάνει έναν soft processor (Altera Nios II), έναν DDR2 SDRAM controller και το RBM Module. Ο soft processor λειτουργεί ως ο ενδιάμεσος μεταξύ του χρήστη και του RBM Module. Είναι υπεύθυνος για την αρχικοποίηση των βαρών, διαβάζει από την SDRAM τους ορατούς νευρώνες, αρχικοποιεί τον αλγόριθμο και επιστρέφει τα αποτελέσματα στον χρήστη. Σε ένα πρώτο, αφηρημένο, επίπεδο το RBM Module περιέχει έναν πίνακα βαρών και νευρώνων που δίνονται ως είσοδοι σε έναν πίνακα πολλαπλασιαστών και έπειτα με την χρήση αθροιστών επιτυγχάνεται ο πολλαπλασιασμός πινάκων. Έπειτα, υπολογίζεται η σιγμοειδής συνάρτηση για να βρεθεί η πιθανότητα ενεργοποίησης ενός νευρώνα και έπειτα επιτυγχάνεται η ενεργοποίηση αυτού με την χρήση ενός συγκριτή και μιας γεννήτριας τυχαίων αριθμών. Έπειτα, συνεχίζεται η επανάληψη με τις εναλλαγές μεταξύ των φάσεων του RBM μέχρι να ικανοποιηθεί η συνθήκη τερματισμού που έχει δοθεί από τον χρήστη. Κάθε βήμα είναι μέρος μια διοχέτευσης (pipeline) και έχουμε throughput της τάξης των 256 πολλαπλασιασμών και προσθέσεων ανά κύκλο. Αυτό έχει να κάνει με τους πολλαπλασιαστές που είναι διαθέσιμοι στην συγκεκριμένη πλακέτα και πόσοι από αυτούς διατίθενται για την σχεδίαση του RBM Module.

Σε μια πιο λεπτομερή περιγραφή το RBM Module χωρίζεται σε μικρότερα τμήματα καθένα από τα οποία διαθέτει έναν πίνακα πολλαπλασιαστών, αθροιστές, ενσωματωμένη μνήμη και λογικά στοιχεία. Τα βάρη και τα δεδομένα διαμοιράζονται σε όλα τα τμήματα και κάθε ένα από αυτά κατέχει ένα μικρό μέρος του δικτύου. Αυτή είναι και η δύναμη της συγκεκριμένης σχεδίασης καθώς επιτυγχάνεται έτσι «τοπική» επικοινωνία και αποφεύγονται καθυστερήσεις. Επιπλέον αυτή η τμηματοποίηση είναι που παρέχει και επεκτασιμότητα καθώς για μεγαλύτερες σχεδιάσεις μπορούμε να προσθέσουμε στιγμιότυπα τέτοιων μικρών τμημάτων. Τα γενικότερα σήματα που χρειάζονται να επικοινωνούνται μεταξύ των τμημάτων μπαίνουν σε buffers.

Μια πιο λεπτομερής απεικόνιση του RBM Module φαίνεται στην παρακάτω εικόνα :



Εικόνα 22 RBM Module [31]

Ο πολλαπλασιασμός πινάκων γίνεται και στις τρείς φάσεις του RBM, κατά την διαδικασία δειγματοληψίας των κρυφών και ορατών νευρώνων και κατά την ανανέωση των βαρών. Για να επιτευχθούν αυτοί οι πολλαπλασιασμοί στην παρούσα υλοποίηση, όπου τα δεδομένα δεν βρίσκονται σε ενσωματωμένα κυκλώματα αλλά σε κάποια εξωτερική μνήμη, πρέπει να λυθεί το πρόβλημα του ανεστραμμένου πίνακα των βαρών. Εδώ οι συγγραφείς ανάλογα με την «φάση» του πολλαπλασιασμού που θέλουμε να εκτελέσουμε χρησιμοποιούν και αντίστοιχη λογική και σχεδίαση. Σύμφωνα με τους συγγραφείς ο πολλαπλασιασμός πινάκων (C = A x B) μπορεί να ερμηνευθεί με τρείς διαφορετικούς :

1. Ως πολλαπλοί γραμμικοί συνδυασμοί διανυσμάτων :

$$\begin{bmatrix} C_{1,j} \\ C_{2,j} \\ \dots \\ C_{m,j} \end{bmatrix} = \sum_{i=1}^{k} \left(B_{i,j} \begin{bmatrix} A_{1,i} \\ A_{2,i} \\ \dots \\ A_{m,i} \end{bmatrix} \right)$$

2. Ως διάνυσμα πολλαπλών εσωτερικών γινομένων :

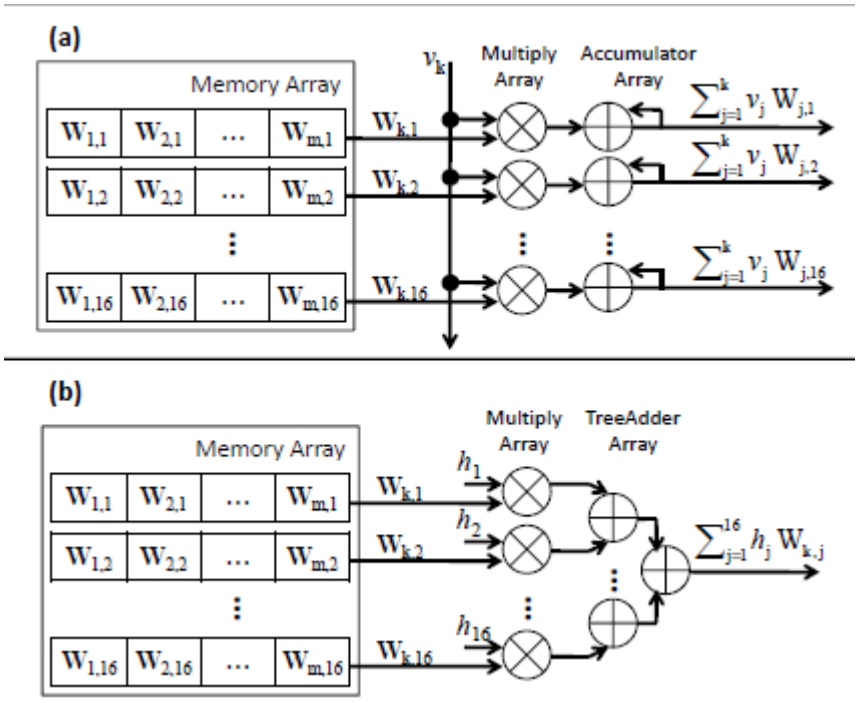
$$C_{i,j} = [A_{i,1} A_{i,2} \dots A_{i,k}] \cdot \begin{bmatrix} B_{1,j} \\ B_{2,j} \\ \dots \\ B_{k,j} \end{bmatrix}$$

3. Ως άθροισμα διανύσματος εξωτερικών γινομένων :

$$C = \sum_{i=1}^{k} \left(\begin{bmatrix} A_{1,i} \\ A_{2,i} \\ \dots \\ A_{m,i} \end{bmatrix} \times [B_{i,1} B_{i,2} \dots B_{i,n}] \right)$$

Έτσι χρησιμοποιούμε αντίστοιχα τους πολλαπλασιασμούς για:
1. Την φάση ανοικοδόμησης
2. Την φάση υπολογισμού των κρυφών μονάδων
3. Την φάση ενημέρωσης των βαρών.

Η σχεδίαση αυτή φαίνεται στην παρακάτω εικόνα :



Εικόνα 23 Πολλαπλασιασμός Πινάκων για υπολογισμό (a) κρυφούς νευρώνες (b) ορατούς νευρώνες [31]

Με την χρήση αυτής της τεχνικής παρέχεται μια επεκτάσιμη μέθοδος για την επίτευξη πολλαπλασιασμού πινάκων χωρίς να χρειάζεται να πραγματοποιήσουμε αναστροφή του πίνακα των βαρών. Η φάση ανανέωσης χρησιμοποιεί το άθροισμα εξωτερικών γινομένων (3) μεταξύ των ορατών και κρυφών νευρώνων. Τα ορατά δεδομένα γίνονται έτσι και αλλιώς broadcast (α) και οι ορατοί νευρώνες μπορούν να δοθούν ως είσοδοι στους πολλαπλασιαστές (b). Αυτό σημαίνει ότι χρησιμοποιώντας την δομή (a) και αντικαθιστώντας τα βάρη με τις τιμές των κρυφών νευρώνων παίρνουμε την φάση ανανέωσης.

4.2 Αποτίμηση – Συμπεράσματα

Οι προσεγγίσεις αυτές καταφέρνουν μεν να μεταφέρουν τις λειτουργίες μιας RBM σε επίπεδο FPGA,έχουν δε πολλούς περιορισμούς και τα συγκριτικά τους αποτελέσματα είναι αμφισβητήσιμα. Αποτελούν σίγουρα τις πιο αξιόλογες προτάσεις για την μεταφορά RBM σε FPGA όμως παρόλο που διατείνονται ότι είναι επεκτάσιμα και πιο γρήγορα αυτό ισχύει μόνο υπό προϋποθέσεις. Συνοψίζοντας τα μειονεκτήματα των προσεγγίσεων και ξεκινώντας από την πρώτη προσέγγιση ήδη από την αρχή της σχεδίασης οι συγγραφείς έχουν ορίσει ότι τα επίπεδα πρέπει να είναι συμμετρικά και ο αριθμός των κόμβων και των δεσμίδων (batches) πρέπει να είναι δύναμη του 2. Αυτό σε συνδυασμό με το ότι χρησιμοποιείται μεγάλος αριθμός BRAMs για τοπική αποθήκευση μας οδηγεί σε πολύ μικρά δίκτυα κάτι που δεν είναι άμεσα εφαρμόσιμο, καθώς όπως είδαμε στο παράδειγμα του DBN ξεκινάμε από δίκτυα με 784 x 500 κόμβους και καταλήγουμε σε 500 x 2000.

Επιπλέον στην συγκεκριμένη προσέγγιση είναι αποδοτική η χρήση του RBM όταν έχουμε δυαδικές τιμές στους κόμβους, κάτι που δεν ισχύει σε όλες τις περιπτώσεις και η σχεδίαση δεν ενδείκνυται για real values στις τιμές των κόμβων. Εκτός αυτού επιλέγεται, αρχικά, μια ντετερμινιστική μέθοδος για την ενεργοποίηση των νευρώνων, κάτι που βοηθάει στην ταχύτητα αλλά δεν αποτελεί σωστή προσέγγιση. Τέλος ακόμα και σε μεταγενέστερες προσπάθειες για επεκτασιμότητα και διόρθωση των ατελειών [30], υπάρχουν προβλήματα στην επικοινωνία μεταξύ των επιπλέον FPGA και θυσιάζεται κομμάτι της ακρίβειας όπως και της ταχύτητας για την δημιουργία δομών ελέγχου και μεταφοράς δεδομένων από πλακέτα σε πλακέτα.

Όσο αφορά στην δεύτερη προσέγγιση υπάρχουν διαφορετικού είδους προβλήματα. Στην περίπτωση αυτή γίνεται μεγάλη χρήση του διαθέσιμου υλικού και βασίζεται στο κατά πόσο «δυνατή» είναι η πλακέτα. Επιπλέον τα δεδομένα αποθηκεύονται σε μια RAM και στις μετρήσεις δεν λαμβάνεται υπόψιν ο χρόνος μεταφοράς αυτών. Υπάρχει ο περιορισμός ότι έχουμε 16 x 16 πολλαπλασιαστές που σημαίνει ότι ο αριθμός των νευρώνων πρέπει να είναι πολλαπλάσιο του 256 αλλιώς δεν είναι αποδοτικό το pipeline. Για χάρη της τμηματοποίησης (portioning) μειώνεται η επεκτασιμότητα και η επίδοση της σχεδίασης .Επιπλέον τα δεδομένα εισέρχονται μέσω JTAG και επειδή ο πίνακας βαρών αποθηκεύεται σε off chip DRAM κάθε τμήμα πρέπει να ταιριάζει στο bus width της DDR2 δηλαδή να δέχεται 256 bits δεδομένων, κάτι που αφήνει μόνο 16 bit ακρίβειας για τα δεδομένα και δεν είναι ευέλικτο.

Περνώντας στην πρόταση για την επέκταση του μοντέλου σε περισσότερα FPGA [32], βλέπουμε εξαρχής ότι το batch size είναι προβληματικό καθώς πρέπει να είναι μεγέθους 16. Αυτό γιατί στην multi-FPGA αρχιτεκτονική δεν μπορεί κάθε πλακέτα να κρατήσει και τον πίνακα βαρών και ολόκληρα τα δεδομένα του batch αν είναι μεγαλύτερο το μέγεθος. Επιπλέον 16 bits δεδομένων βαρών είναι δυνατόν να μεταφέρονται ανά κύκλο με βάση το interface της DDR2 μνήμης που προείπαμε. Η τοπολογία που προτείνεται είναι point to point και μάλιστα RING και αυτό θέτει περιορισμό στον αριθμό των πλακετών που μπορούν να χρησιμοποιηθούν καθώς εξαρτάται από το πόσα pins έχουν διαθέσιμα για αυτόν τον σκοπό. Επιπλέον σε κάθε FPGA μπορούν να χωρέσουν δίκτυα μόνο 256x256 εξαιτίας του περιορισμού της on chip μνήμης και έτσι χάνεται η ευελιξία που θα μπορούσε να έχει το μοντέλο. Τέλος στην σχεδίαση ισχύει γενικά ότι για n νευρώνες έχουνε O(n^2) πίνακα βαρών που πρέπει να γίνεται stream από μια off chip memory.

Κλείνοντας, είναι πολύ σημαντικό να αναφέρουμε πως δεν υπάρχουν ξεκάθαρα συγκρίσιμα μεγέθη καθώς κάθε υλοποίηση γίνεται με περιορισμούς και συγκρίνεται με custom αλγόριθμους είτε σε Matlab είτε σε C/C++ που τρέχουν σε παλαιού τύπου υπολογιστές με περιορισμένους πόρους. Επιπλέον η σύγκριση γίνεται για κομμάτια συγκεκριμένα υπολογισμών και δεν ισχύουν για όλες τι δυνατές υλοποιήσεις RBM σε software.

4.3 Προτεινόμενη προσέγγιση

Με βάση τα παραπάνω καταλαβαίνουμε ότι υπάρχουν πολλά κενά και είμαστε ακόμη μακριά από μια σταθερή υλοποίηση που θα μπορεί να εξυπηρετήσει αλγορίθμους RBM σε ένα ευρύ

φάσμα εφαρμογών. Είτε λόγω περιορισμών είτε λόγω σχεδιάσεων φαίνεται πως κάτι τέτοιο είναι ακόμα ανέφικτο και παρά την όποια επιτάχυνση επιτυγχάνεται σε επίπεδο συγκεκριμένων υπολογισμών, οι υλοποιήσεις αυτές παραμένουν ακατάλληλες για την άμεση χρησιμοποίησή τους σε εφαρμογές DBN.

Σκοπός της παρούσης εργασίας είναι η προσπάθεια εκμετάλλευσης της παραπάνω γνώσης για την δημιουργία ενός πλαισίου RBM παραμετροποιήσιμου με χρήση Software - Hardware τεχνικών που θα είναι δυνατό να χρησιμοποιηθεί ανάλογα με το ζητούμενο πρόβλημα. Σε αρχικό στάδιο δεν επιζητούμε τόσο την επιτάχυνση, αν και με σωστή σχεδίαση κομμάτι των υπολογισμών θα επωφεληθεί έστω και στο ελάχιστο, όσο την εγκαθίδρυση μιας βάσης πάνω στην οποία θα μπορούμε να χτίσουμε περαιτέρω για την δημιουργία ενός αποδοτικού συστήματος που χρησιμοποιεί RBMs. Αυτό με την σειρά του έχει διαφορετικούς περιορισμούς, κυρίως λόγο απαιτήσεων μεγάλης επεξεργαστικής ισχύς και μνήμης, αλλά έχει το πλεονέκτημα ότι ερευνητές που δεν έχουν εξειδικευμένες γνώσεις hardware θα μπορέσουν να το χρησιμοποιήσουν με σκοπό την επιτάχυνση της διαδικασίας εκπαίδευσης του εκάστοτε σχεδιασμού. Θα χρησιμοποιήσουμε την μεθοδολογία FPGA in the Loop ώστε να επιτύχουμε την δημιουργία ενός Hardware – Software Co-Simulation μοντέλου, σχεδιασμένου κατά κύριο λόγο σε γραφικό περιβάλλον, που θα περιέχει μικρά τμήματα κώδικα και γραφικές αναπαραστάσεις των λειτουργιών σε επίπεδο Hardware – FPGA. Η μεθοδολογία FPGA in the Loop μας επιτρέπει την σύνδεση μιας FPGA πλακέτας στον υπολογιστή και την επικοινωνία της με το software που θέλουμε να βελτιστοποιήσουμε. Μέσω της τεχνικής FIL μπορούμε να μεταφέρουμε κομμάτι υπολογισμών από το software σε επίπεδο hardware. Στόχος μας είναι να χρησιμοποιήσουμε την τεχνική αυτή ώστε να κρατήσουμε την επιτάχυνση που μας δίνει η εκτέλεση των υπολογισμών σε hardware (FPGA), προσπαθώντας παράλληλα να κάνουμε καλύτερη και πιο εύκολη την γενίκευση του μοντέλου για διαφορετικού τύπου RBMs χρησιμοποιώντας το software. Αυτό θα επιτευχθεί με την χρήση λογισμικού Matlab – Simulink και παρά τους όποιους περιορισμούς και ατέλειες, που θα δούμε αναλυτικότερα παρακάτω, αποτελεί μια πρώτη προσπάθεια ενός πλαισίου παραμετροποιήσιμου το οποίο θα μπορεί να βελτιωθεί σε βάθος χρόνου είτε σχεδιαστικά, ξεπερνώντας κάποιες δυσκολίες, είτε επεξεργαστικά καθώς εισέρχονται νέες τεχνολογίες, και σε επίπεδο FPGA boards και σε επίπεδο λογισμικού και ηλεκτρονικών υπολογιστών, που μπορούν να επιταχύνουν του υπολογισμούς και την απόδοση του μοντέλου.

Πριν περάσουμε στην ανάλυση και σχεδίαση του συστήματος αξίζει να αναφέρουμε ότι θα βασιστούμε σε πραγματικό κώδικα για handwritten digits recognition σε Matlab όπως παρέχεται από τους δημιουργούς του (Hinton & Salakhutdinov) και σε πραγματικά δεδομένα που αποτελούν συγκρίσιμο μέγεθος (MNIST Database for Handwritten digits recognition).

5. Παρουσίαση μοντέλου

Στο παρόν κεφάλαιο θα παρουσιάσουμε βήμα – βήμα την σχεδίαση ενός RBM, ξεκινώντας από τον Matlab κώδικα μέχρι και την τελική FPGA In the Loop υλοποίηση που θα περιλαμβάνει τις RBM λειτουργίες διαμοιρασμένες σε Simulink software και FPGA. Η όλη διαδικασία περιλαμβάνει αρκετά στάδια και υπάρχουν περιορισμοί, κυρίως λόγο εργαλείων. Θα ξεκινήσουμε από την ανάλυση του κώδικα και των δεδομένων και θα δείξουμε μια εκτέλεση μέρους του κώδικα σε Matlab. Έπειτα θα σπάσουμε των κώδικα σε μικρότερα κομμάτια και θα σχεδιάσουμε blocks στο Simulink περιβάλλον που να εκτελούν αυτές τις λειτουργίες. Τέλος έχοντας αυτά τα blocks στο Simulink θα χρησιμοποιήσουμε τον Xilinx System Generator για να μεταφέρουμε κομμάτια υπολογισμών από τα blocks αυτά σε επίπεδο FPGA. Στο τέλος του κεφαλαίου θα κάνουμε μια αποτίμηση της προσπάθειας, θα εξηγήσουμε τα προβλήματα που προκύπτουν και θα προτείνουμε τρόπους για να ξεπεραστούν και να βελτιωθεί το μοντέλο.

5.1 Κώδικας και Data Set

Στο [33] δίνεται από του δημιουργούς ο κώδικας,σε Matlab, για την δημιουργία και εκπαίδευση του DBN που είδαμε προηγουμένως και χρησιμοποιείται για την αναγνώριση ψηφίων γραμμένων με το χέρι. Στο [34] δίνεται το MNIST database που περιέχει τα 60,000 δεδομένα προς εκπαίδευση και 10,000 δεδομένα ελέγχου για το μοντέλο και αποτελεί υποσύνολο ενός ακόμη μεγαλύτερου dataset. Επίσης δίνονται οδηγίες για τον τρόπο αποθήκευσης και προσπέλασης των αρχείων. Όσο αφορά τον κώδικα θα παρουσιάσουμε κομμάτια του για να κατανοήσουμε την λειτουργία του DBN,να εξηγήσουμε τα σημεία που είναι σημαντικά και να αρχίσουμε να βλέπουμε πως θα γίνει η μεταφορά τους σε Simulink.

Ο κώδικας είναι γραμμένος σε Matlab και αποθηκευμένος σε .m αρχεία. Τα επιμέρους αυτά αρχεία καλούνται από ένα βασικό .m αρχείο ανάλογα με την φάση εκτέλεσης που βρισκόμαστε.

Το κύριο αρχείο εκτέλεσης στην περίπτωσή μας είναι το mnistclassify.m :

```
20  clear all
21  close all
22
23  maxepoch=50;
24  numhid=500; numpen=500; numpen2=2000;
25
26  fprintf(1,'Converting Raw files into Matlab format \n');
27  converter;
28
29  fprintf(1,'Pretraining a deep autoencoder. \n');
30  fprintf(1,'The Science paper used 50 epochs. This uses %3i \n', maxepoch);
31
32  makebatches;
33  [numcases numdims numbatches]=size(batchdata);
34
35  fprintf(1,'Pretraining Layer 1 with RBM: %d-%d \n',numdims,numhid);
36  restart=1;
37  rbm;
38  hidrecbiases=hidbiases;
39  save mnistvhclassify vishid hidrecbiases visbiases;
40
41  fprintf(1,'\nPretraining Layer 2 with RBM: %d-%d \n',numhid,numpen);
42  batchdata=batchposhidprobs;
43  numhid=numpen;
44  restart=1;
45  rbm;
46  hidpen=vishid; penrecbiases=hidbiases; hidgenbiases=visbiases;
47  save mnisthpclassify hidpen penrecbiases hidgenbiases;
48
49  fprintf(1,'\nPretraining Layer 3 with RBM: %d-%d \n',numpen,numpen2);
50  batchdata=batchposhidprobs;
51  numhid=numpen2;
52  restart=1;
53  rbm;
54  hidpen2=vishid; penrecbiases2=hidbiases; hidgenbiases2=visbiases;
55  save mnisthp2classify hidpen2 penrecbiases2 hidgenbiases2;
56
57  backpropclassify;
```

Εικόνα 24 Κύριο αρχείο που καλεί τα υπόλοιπα

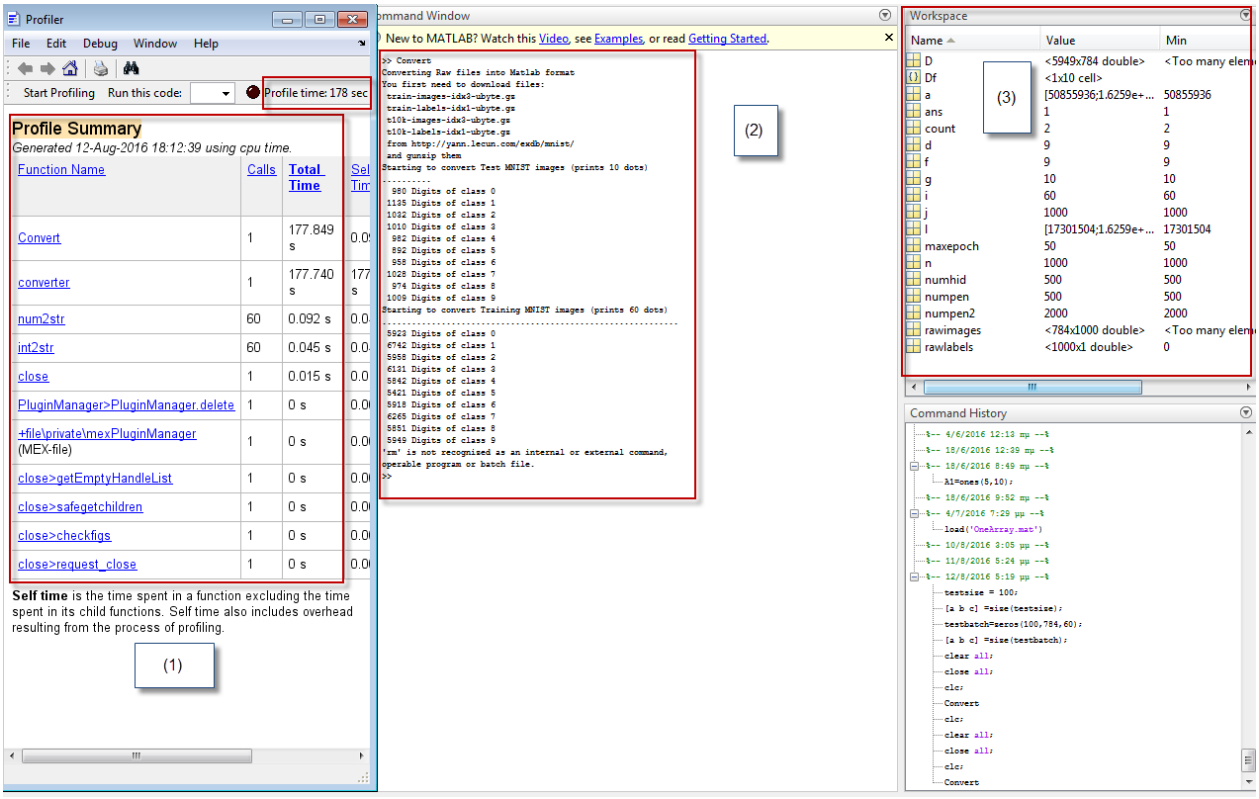
Δεν θα επεκταθούμε στην ανάλυση κάθε γραμμής κώδικα, παραθέτουμε όμως κομμάτια από τα .m αρχεία για να έχει ο αναγνώστης ένα σημείο αναφοράς κατά την ανάλυση της λειτουργίας του κώδικα.

Όπως βλέπουμε στην προηγούμενη εικόνα ορίζεται εξαρχής ο αριθμός των κρυφών νευρώνων που θα περιέχει κάθε επίπεδο RBM (500,500,2000) και αριθμός τον εποχών εκπαίδευσης (50). Έπειτα καλείται το converter αρχείο που αναλαμβάνει να μετατρέψει τα raw αρχεία των

δεδομένων σε μορφή κατάλληλη για εισαγωγή στο Matlab,πρώτα σε ascii και μετά σε mat files, χωρισμένα σε κλάσεις ψηφίων για training και testing.

Έπειτα καλείται το αρχείο makebatches το οποίο καθορίζει τον αριθμό των δεσμίδων και το μέγεθος αυτών και τοποθετεί σε random σειρά τα δεδομένα μέσα σε αυτά. Η έξοδος από αυτό το αρχείο είναι επίσης ο αριθμός των ορατών νευρώνων (numdims),ο οποίος είναι 784 (28x28 pixels). Στην συνέχεια καλείται ο αλγόριθμος που τρέχει το RBM από το αρχείο rbm .Έχει ως παραμέτρους το μέγεθος και τον αριθμό των δεσμίδων (batches) καθώς επίσης και τον αριθμό των ορατών και κρυφών νευρώνων. Ο αλγόριθμος αυτός καλείται άλλες 2 φορές για να λειτουργήσει το DBN με την διαφορά ότι αλλάζει ο αριθμός των κρυφών νευρώνων ανάλογα σε ποιο επίπεδο RBM βρισκόμαστε και επιπλέον τα βάρη μεταφέρονται από το προηγούμενο επίπεδο. Τέλος καλείται το αρχείο backpropclassify που εκτελεί την οπισθοδιάδοση για την βελτιστοποίηση των βαρών του τελευταίου επιπέδου,ώστε να επιτευχθεί καλύτερη ταξινόμηση.

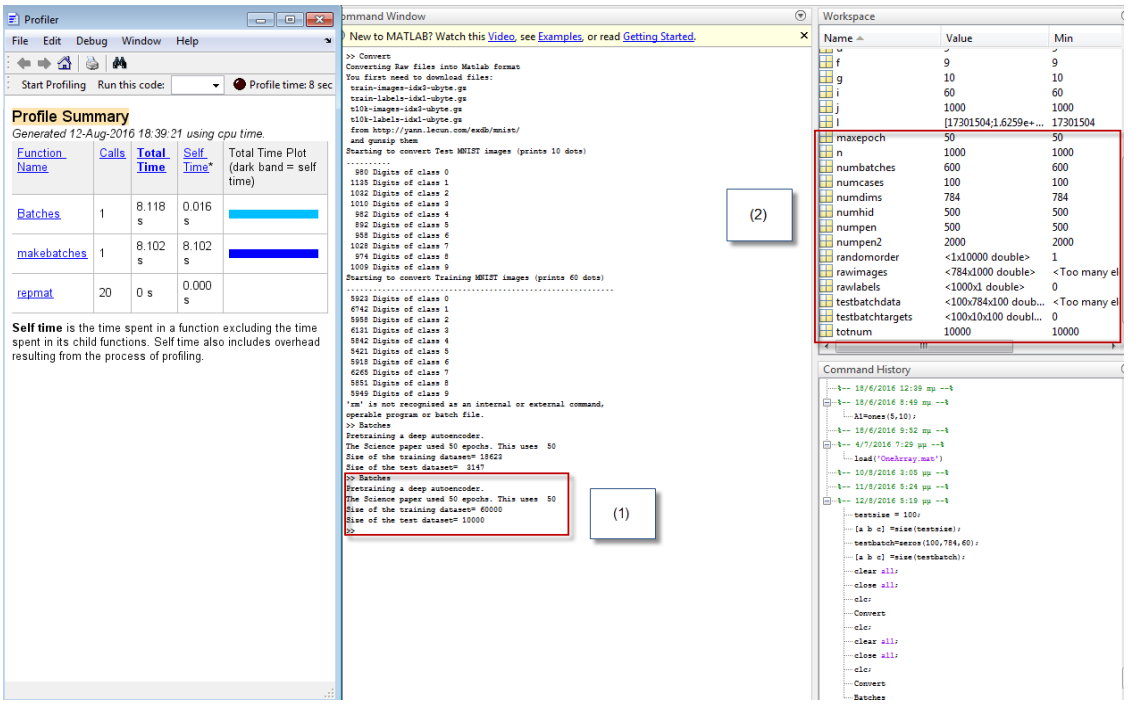
Ακολουθεί το παράδειγμα εκτέλεσης σε Matlab, της προπαρασκευής δεδομένων (converter.m) και της δημιουργίας των δεσμίδων (makebatches.m) . Για την καλύτερη παρακολούθηση των εκτελέσεων του κώδικα και των αποτελεσμάτων αυτού, τον εκτελούμε βήμα προς βήμα.



Εικόνα 25 Convert των δεδομένων

Αφού εκτελέσαμε το πρώτο κομμάτι του κώδικα βλέπουμε ότι (1) ο Profiler του Matlab μας δίνει τα στατιστικά του χρόνου εκτέλεσης του κώδικα, (2) στο command window μας δείχνει αναλυτικά των αριθμό των training και test images (έξοδος του κώδικα) και (3) στο workspace έχουν ήδη αποθηκευτεί οι πρώτες μεταβλητές.

Στην συνέχεια εκτελούμε το κομμάτι που αρχικοποιεί το μέγεθος και τον αριθμό των δεσμίδων με βάση τον αριθμό των δεδομένων μας. Αυτό φαίνεται και στην παρακάτω εικόνα.



Εικόνα 26 Μέγεθος και αριθμός δεσμίδων (batches)

Παρακάτω παραθέτουμε τον κώδικα του rbm.m καθώς είναι το πιο σημαντικό κομμάτι της υλοποίησης και τις λειτουργίες του θα μεταφέρουμε στην συνέχεια σε επίπεδο Simulink και FPGA. Όπως βλέπουμε λοιπόν στις εικόνες 27-28 υπάρχουν αρκετές μεταβλητές που επηρεάζουν την εκπαίδευση ενός RBM. Οι μεταβλητές epsilon αποτελούν τους ρυθμούς εκπαίδευσης,το weightcost και οι μεταβλητές momentum παίζουν ρόλο στον τρόπο ενημέρωσης των βαρών και βοηθούν στην καλύτερη εκπαίδευση του RBM. Στο [16] δίνεται ένας αναλυτικός οδηγός όλων των παραμέτρων που πρέπει να ληφθούν υπόψιν όταν εκπαιδεύουμε ένα RBM και ο συγγραφέας προσφέρει επιπλέον συμβουλές από την πραγματική του εμπειρία. Στην συνέχεια έχουμε αρχικοποίηση των συμμετρικών βαρών και των biases σε τυχαίες τιμές (αν είναι η πρώτη φορά εκτέλεσης του αλγορίθμου).

Έπειτα ξεκινάει ο κύριος αλγόριθμος του RBM και διακρίνεται σε τρείς φάσεις: στην Positive phase, στην Negative phase και στην Update phase. Για κάθε εποχή και κάθε δεσμίδα η πρώτη φάση είναι η δημιουργία του κρυφού επίπεδου (Generate) ακολουθούμενη από την negative (Reconstruct) φάση όπου έχουμε την δημιουργία του ορατού επιπέδου με παράλληλο υπολογισμό του επόμενου κρυφού επιπέδου (Generate). Αυτό συνιστά Contrastive Divergence ενός βήματος. Υπολογίζεται το σφάλμα και ξεκινάει η φάση ανανέωσης των συμμετρικών βαρών και των biases, Στο τέλος κάθε εποχής εμφανίζεται το σφάλμα που μας δείχνει κατά πόσο πάει καλά η εκπαίδευσή μας.

Όταν έχουν τελειώσει όλες οι επαναλήψεις θα σώσουμε τις τιμές των βαρών και των biases (τις χρειαζόμαστε για το fine tuning του DBN) και θα προχωρήσουμε στην εκπαίδευση του επόμενου επιπέδου RBM έχοντας ως είσοδο τους κρυφούς νευρώνες που πήραμε από το προηγούμενο επίπεδο.

Ακολουθεί ο κώδικας του RBM σε δύο εικόνες :

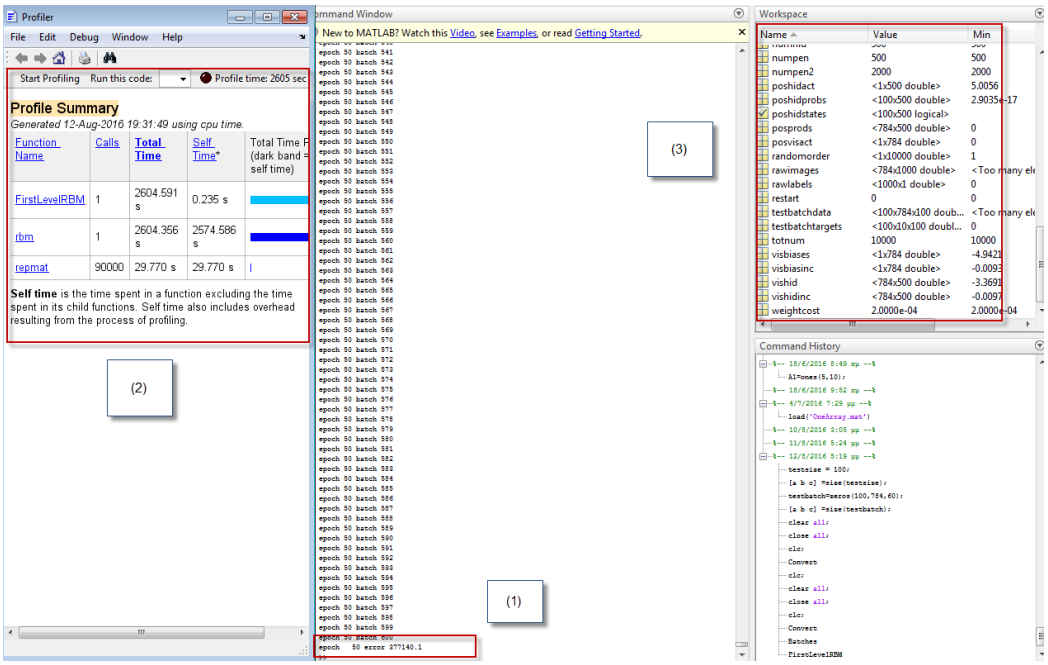
```
epsilonvb      = 0.1;   % Learning rate for biases of visible units
epsilonhb      = 0.1;   % Learning rate for biases of hidden units
weightcost  = 0.0002;
initialmomentum  = 0.5;
finalmomentum    = 0.9;

[numcases numdims numbatches]=size(batchdata);

if restart ==1,
  restart=0;
  epoch=1;

% Initializing symmetric weights and biases.
  vishid     = 0.1*randn(numdims, numhid); %(W)
  hidbiases  = zeros(1,numhid);
  visbiases  = zeros(1,numdims);

  poshidprobs = zeros(numcases,numhid);
  neghidprobs = zeros(numcases,numhid);
  posprods    = zeros(numdims,numhid);
  negprods    = zeros(numdims,numhid);
  vishidinc  = zeros(numdims,numhid);
  hidbiasinc = zeros(1,numhid);
  visbiasinc = zeros(1,numdims);
  batchposhidprobs=zeros(numcases,numhid,numbatches);
end

for epoch = epoch:maxepoch,
 fprintf(1,'epoch %d\r',epoch);
 errsum=0;
 for batch = 1:numbatches,
 fprintf(1,'epoch %d batch %d\r',epoch,batch);
%%%%%%%%%% START POSITIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  data = batchdata(:,:,batch); %Visible units V
  poshidprobs = 1./(1 + exp(-data*vishid - repmat(hidbiases,numcases,1))); %Updating hidden states (ph)
  batchposhidprobs(:,:,batch)=poshidprobs; %Setting hidden layer states for every batch
  posprods    = data' * poshidprobs; % V*H Data
  poshidact   = sum(poshidprobs);
  posvisact = sum(data);

%%%%%%%%%% END OF POSITIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  poshidstates = poshidprobs > rand(numcases,numhid); % Firing up neurons in hidden layer

%%%%%%%%%% START NEGATIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  negdata = 1./(1 + exp(-poshidstates*vishid' - repmat(visbiases,numcases,1))); % Updating Visible states (pv)
  neghidprobs = 1./(1 + exp(-negdata*vishid - repmat(hidbiases,numcases,1))); % Updating hdden states reconstruction
  negprods  = negdata'*neghidprobs; % V*H Model
  neghidact = sum(neghidprobs);
  negvisact = sum(negdata);
%%%%%%%%%% END OF NEGATIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Εικόνα 27 RBM Code part1

```
  err= sum(sum( (data-negdata).^2 ));
  errsum = err + errsum;

   if epoch>5,
     momentum=finalmomentum;
   else
     momentum=initialmomentum;
   end;

%%%%%%%%%% UPDATE WEIGHTS AND BIASES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    vishidinc = momentum*vishidinc + ...
                epsilonw*( (posprods-negprods)/numcases - weightcost*vishid); % m*W + E*(CD DW - weight*W))
    visbiasinc = momentum*visbiasinc + (epsilonvb/numcases)*(posvisact-negvisact);
    hidbiasinc = momentum*hidbiasinc + (epsilonhb/numcases)*(poshidact-neghidact);

    vishid = vishid + vishidinc; % Accumulating waight Update
    visbiases = visbiases + visbiasinc; % Visible Bias Update
    hidbiases = hidbiases + hidbiasinc; % Hidden Bias Update

%%%%%%%%%%%%%%%%%% END OF UPDATES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

  end
  fprintf(1, 'epoch %4i error %6.1f  \n', epoch, errsum);
end;
```

Εικόνα 28 RBM Code part2

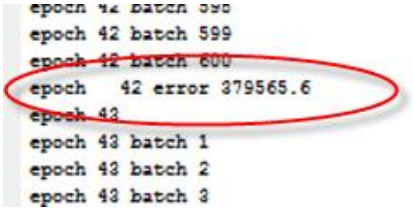
Παράδειγμα εκτέλεσης του κώδικα φαίνεται παρακάτω :



Εικόνα 29 Εκτέλεση RBM σε Matlab

Στην παραπάνω εικόνα βλέπουμε πως (1) το RBM έτρεξε για 50 εποχές δίνοντας μας το σφάλμα που έχει τιμή 377140.1, (2) στον Profiler ήδη βλέπουμε ότι αυτή η εκτέλεση χρειάστηκε σημαντικό χρόνο (2605 sec δηλαδή περίπου 43 με 44 mins) και (3) το workspace μας πλέον περιέχει όλες τις μεταβλητές που πήραν μέρος στου υπολογισμούς.

Είπαμε πως για να δούμε αν δουλεύει σωστά η εκπαίδευσή μας πρέπει το σφάλμα να μειώνεται κάτι που επιβεβαιώνεται σε κάθε εποχή όπου αν πάμε λίγο πιο πάνω στο command window του Matlab μπορούμε να πάρουμε το σφάλμα για μια προηγούμενη εποχή. Αυτό φαίνεται στην παρακάτω εικόνα :



Εικόνα 30 Σφάλμα εποχής 42

5.2 Ανάπτυξη του μοντέλου σε Matlab Simulink

Τα επόμενα βήματα περιλαμβάνουν την μεταφορά του κώδικά μας σε περιβάλλον Simulink και έπειτα την επανασχεδίαση του μοντέλου ώστε να μπορούν να κατέβουν σε FPGA κομμάτια κώδικα που αφορούν το RBM μας.

5.2.1 Περιβάλλον Simulink

Πριν προχωρήσουμε στην παρουσίαση των βημάτων της σχεδίασης καλό θα ήταν να περιγράψουμε το περιβάλλον του Matlab Simulink και να πούμε δυο λόγια για τις λειτουργίες του. Στον πυρήνα του το Simulink είναι ένα γραφικό προγραμματιστικό περιβάλλον για μοντελοποίηση, ανάλυση και προσομοίωση δυναμικών συστημάτων [35],[36]. Παρέχει block διαγράμματα και έτοιμες βιβλιοθήκες για την μοντελοποίηση και προσομοίωση συστημάτων

από διάφορα πεδία, και είναι ενσωματωμένο μες στο Matlab περιβάλλον. Αυτό σημαίνει ότι εναρμονίζει τις λειτουργίες του με το Matlab και μπορεί να παράγει κώδικα για αυτό, να χρησιμοποιήσει κώδικα και αρχεία Matlab καθώς επίσης να καλείται μέσα από αυτό [37]. Ένα απλό παράδειγμα σχεδίασης Simulink μπορούμε να δούμε στο Παράρτημα Α.

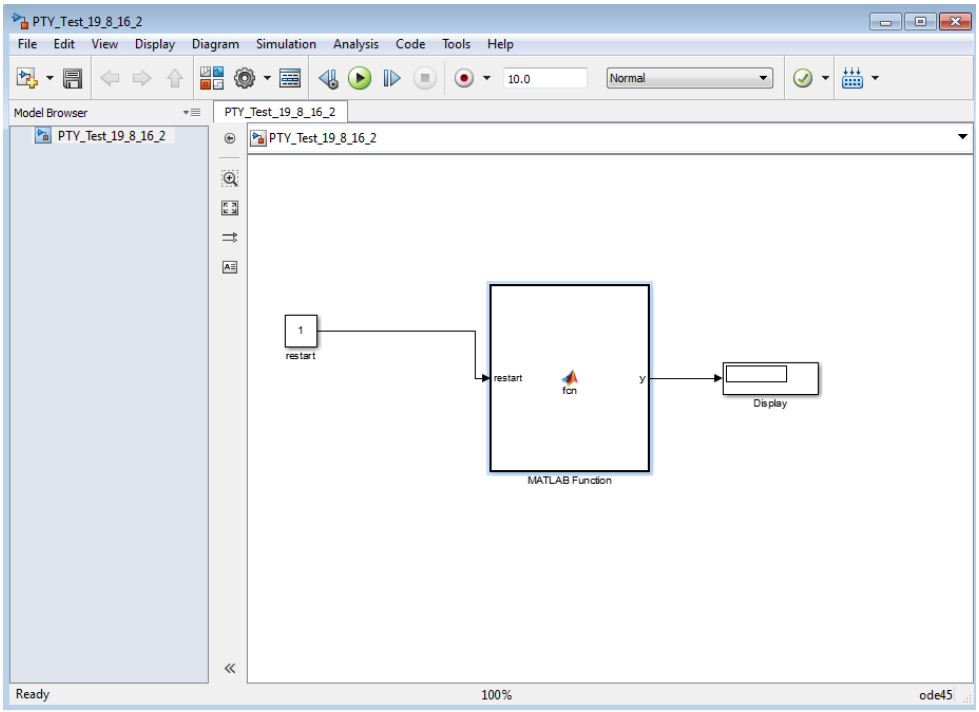
Όλα τα μπλοκ, που μπορούμε να χρησιμοποιήσουμε, έχουν πλήθος παραμέτρων που μπορούν να μεταβληθούν ανάλογα με την λειτουργία που θέλουμε να εκτελέσουμε. Το εργαλείο μας επιτρέπει μεγάλη ελευθερία ώστε να μπορούμε να καθορίσουμε τον τύπο της προσομοίωσης (Time based - Sample based),των τύπο των σημάτων (Continuous – Discrete) και ακόμα την είσοδο και την έξοδο του μοντέλου καθώς επίσης παρέχει έτοιμα block για στοχευμένες – εξειδικευμένες λειτουργίες από πολλά πεδία επιστημών.

Όπως καταλαβαίνουμε σε πολύ μεγάλα μοντέλα μπορούμε να συναντήσουμε πλήθος αλληλεπιδράσεων μεταξύ μπλοκ που μπορεί να δέχονται δεδομένα από διαφορετικές πηγές (αρχεία, Matlab Workspace, Real Time Streaming από sensors) να εκτελούν πολύπλοκες λειτουργίες, ακόμα και κώδικα γραμμένου σε Matlab, και να παρέχουν αποτελέσματα σε περαιτέρω συστήματα (είτε απευθείας, είτε αποθηκεύοντας σε αρχεία, είτε γυρίζοντας τα πίσω στο Matlab περιβάλλον).

Στην περίπτωσή μας, επειδή η φύση του προβλήματος μας είναι αρκετά διαφορετική από την συνήθης λειτουργία του Simulink (σήματα – επεξεργασία- έξοδος),υπάρχουν κάποιοι περιορισμοί που πρέπει να ξεπεραστούν καθώς επίσης και κάποια μειονεκτήματα σε ότι αφορά στην σχεδίαση. Αυτά θα συζητούνται κατά την διάρκεια της παρουσίασης της σχεδίασης ανάλογα με το βήμα που βρισκόμαστε.

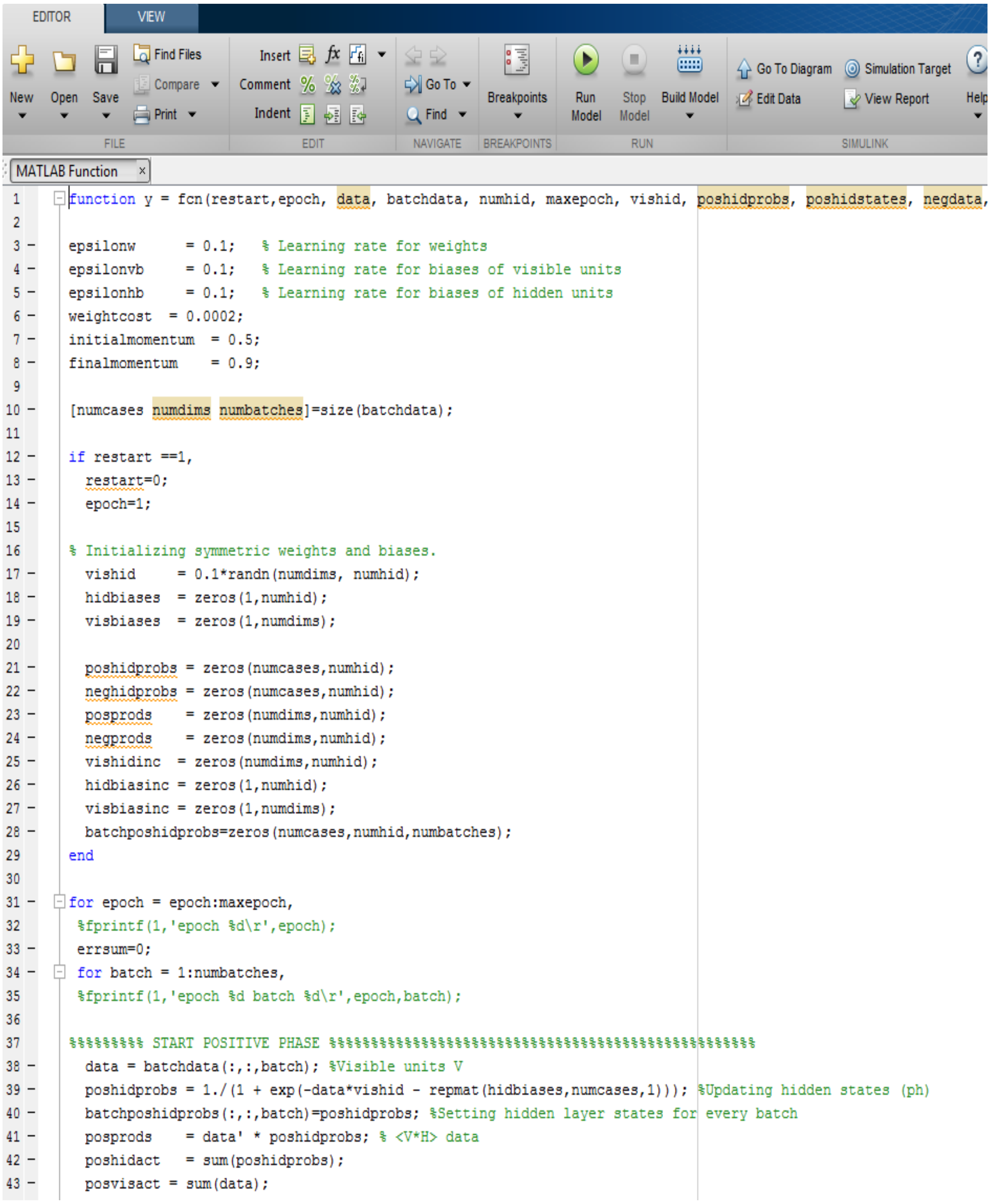
5.2.2 Σχεδίαση RBM σε Simulink

Μια απλή προσέγγιση, για να ξεκινήσουμε την σχεδίαση, είναι να μεταφέρουμε των κώδικα μας σε ένα block σε περιβάλλον Simulink. Αυτό το βήμα μας δίνει μια πρώτη εικόνα του τι μπορούμε να περιμένουμε από την σχεδίαση καθώς επίσης μας παρουσιάζει τους πρώτους περιορισμούς και τις ιδιαιτερότητες αυτής. Τοποθετώντας τον κώδικα μας σε ένα custom Matlab Function block μια απλή σχεδίαση θα έχει την παρακάτω εικόνα :



Εικόνα 31 Μια απλή προσέγγιση της σχεδίασης

Στην συγκεκριμένη περίπτωση το restart μας δείχνει σε πιο στάδιο της εκπαίδευσης είμαστε (αρχικό) και το y αποτελεί την έξοδο του block που έχουμε αντιστοιχίσει στο σφάλμα. Ο κώδικας που εκτελείται μες το Matlab Function block είναι για αρχή, ο ίδιος κώδικας του RBM που περιγράψαμε προηγουμένως (5.1). Ένα κομμάτι του μπορούμε να δούμε ανοίγοντας με διπλό κλικ το block :




```
function y = fcn(restart,epoch, data, batchdata, numhid, maxepoch, vishid, poshidprobs, poshidstates, negdata,

epsilonw      = 0.1;   % Learning rate for weights
epsilonvb     = 0.1;   % Learning rate for biases of visible units
epsilonhb     = 0.1;   % Learning rate for biases of hidden units
weightcost  = 0.0002;
initialmomentum  = 0.5;
finalmomentum    = 0.9;

[numcases numdims numbatches]=size(batchdata);

if restart ==1,
  restart=0;
  epoch=1;

% Initializing symmetric weights and biases.
  vishid     = 0.1*randn(numdims, numhid);
  hidbiases  = zeros(1,numhid);
  visbiases  = zeros(1,numdims);

  poshidprobs = zeros(numcases,numhid);
  neghidprobs = zeros(numcases,numhid);
  posprods    = zeros(numdims,numhid);
  negprods    = zeros(numdims,numhid);
  vishidinc  = zeros(numdims,numhid);
  hidbiasinc = zeros(1,numhid);
  visbiasinc = zeros(1,numdims);
  batchposhidprobs=zeros(numcases,numhid,numbatches);
end

for epoch = epoch:maxepoch,
 %fprintf(1,'epoch %d\r',epoch);
 errsum=0;
 for batch = 1:numbatches,
 %fprintf(1,'epoch %d batch %d\r',epoch,batch);

%%%%%%%%% START POSITIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  data = batchdata(:,:,batch); %Visible units V
  poshidprobs = 1./(1 + exp(-data*vishid - repmat(hidbiases,numcases,1))); %Updating hidden states (ph)
  batchposhidprobs(:,:,batch)=poshidprobs; %Setting hidden layer states for every batch
  posprods    = data' * poshidprobs; % <V*H> data
  poshidact   = sum(poshidprobs);
  posvisact = sum(data);
```

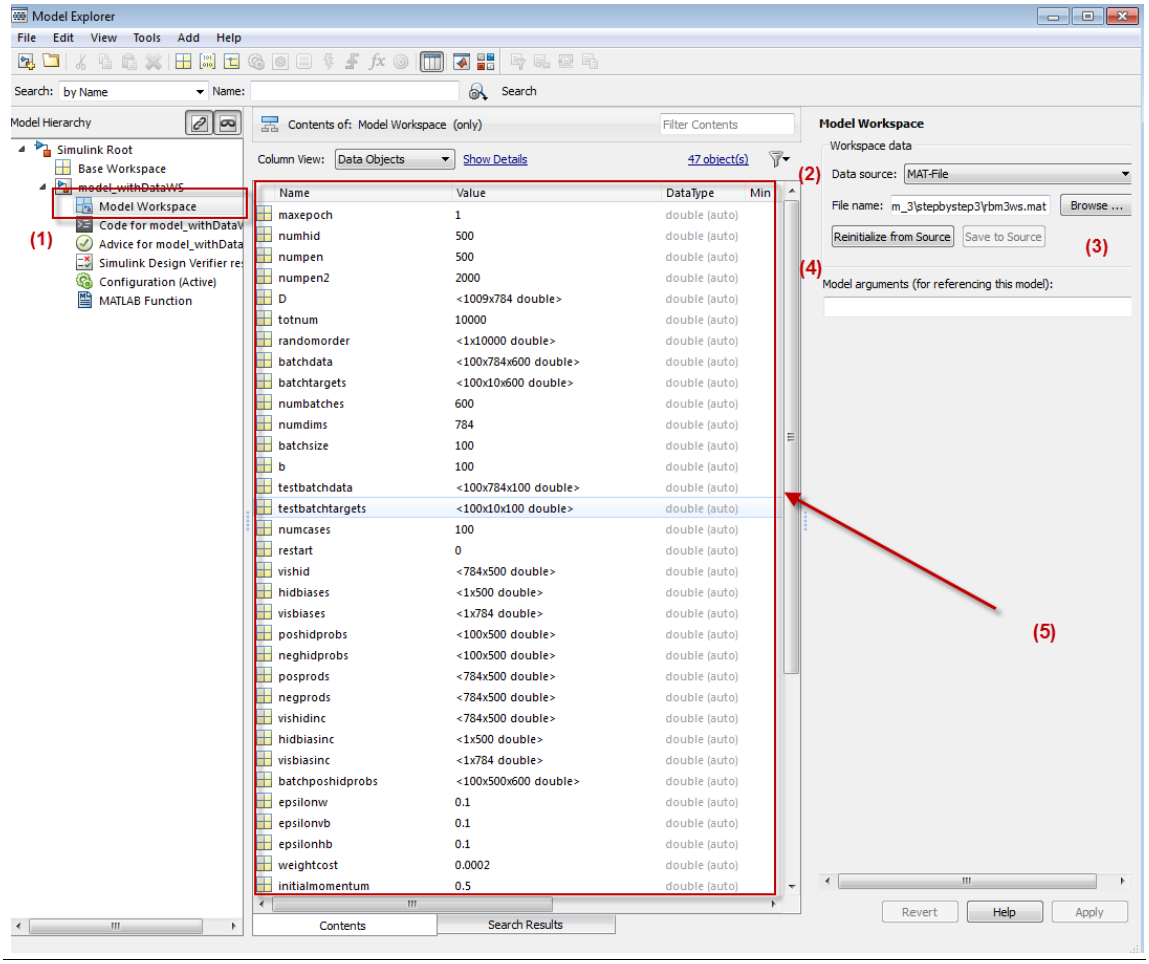
Εικόνα 32 Εικόνα μέρους του κώδικα που εκτελείται στο block

Σε αυτό το σημείο βλέπουμε ότι έχουμε καταφέρει να εντάξουμε τον κώδικα Matlab του RBM που έχουμε, σε ένα μοντέλο Simulink και έχουμε αντιστοιχίσει την έξοδο του σφάλματος σε ένα display block ώστε να μπορούμε να καταλάβουμε αν η εκπαίδευση μας λειτουργεί κανονικά. Ακόμα όμως δεν έχουμε μιλήσει για τα δεδομένα εισόδου μας. Πως θα μπορέσουμε να εισάγουμε τα δεδομένα μας σε ένα τέτοιο μοντέλο και επιπλέον πως θα ορίσουμε τις μεταβλητές που λειτουργούν πάνω σε αυτά ; Κάπου εδώ αρχίζουν να εμφανίζονται οι ιδιαιτερότητες της σχεδίασής μας. Αρχικά πρέπει να έχουμε ήδη τρέξει τα αρχεία που μετασχηματίζουν τα δεδομένα μας και ορίζουν τον αριθμό και το μέγεθος των δεσμίδων προς εκπαίδευση. Επιπλέον για να μεταφέρουμε τα δεδομένα αυτά από το Matlab στο Simulink, και αντιστρόφως, υπάρχουν δύο επιλογές. Η πρώτη είναι η χρησιμοποίηση των From και To Workspace block και η δεύτερη είναι να αποθηκεύσουμε το Workspace του Matlab σε ένα αρχείο .mat και να χρησιμοποιήσουμε αυτό ως είσοδο στο Simulink.

Η πρώτη επιλογή δεν είναι λειτουργική στην περίπτωση του μοντέλου που θέλουμε να σχεδιάσουμε καθώς, εξ' ορισμού τα δύο αυτά μπλοκ περιέχουν την έννοια του χρόνου. Θεωρούνται ως μπλοκ που μεταφέρουν δεδομένα μεταξύ προσομοιώσεων(simin και simout) και απαιτούν ένα timestamp για τα δεδομένα που μεταφέρονται μέσω αυτών. Έτσι δεν καλύπτεται αυτή απαίτηση από τα δεδομένα μας, οι υπολογισμοί μας δεν γίνονται με κάποιο time step ούτε έχουν να κάνουν με σήματα που έχουν χρόνους εκπομπής, και ακόμα και αν ήταν εφικτό να δώσουμε με κάποιο τρόπο χρονική διάσταση στα δεδομένα μας,τότε ο όγκος των δεδομένων προς επεξεργασία μας θα αυξανόταν δραματικά που, όπως θα δούμε και παρακάτω, είναι κάτι απαγορευτικό για το μοντέλο μας.

Η δεύτερη επιλογή είναι η κατάλληλη καθώς, εκτός των προηγουμένων, τα δεδομένα μας αποτελούνται ως επί το πλείστον από δισδιάστατους και τρισδιάστατους πίνακες και επιπλέον στην συνέχεια θα δούμε ότι για την περίπτωση διαφορετικών παραμέτρων για διαφορετικές υλοποιήσεις RBM καλό θα ήταν να αρχικοποιείται το μοντέλο με το εκάστοτε .mat αρχείο. Έκτος αυτού όμως έχουμε να λύσουμε ένα επιπλέον πρόβλημα που έχει να κάνει με τον ορισμό των μεταβλητών που χρησιμοποιούνται στα Simulink blocks. Για να χρησιμοποιηθεί κώδικας σε αυτά τα block πρέπει να έχουν ορισθεί οι μεταβλητές που περιέχει. Αυτό γίνεται με μια διαδικασία που συσχετίζει τα δεδομένα που φορτώνουμε στο μοντέλο με τις μεταβλητές που χρησιμοποιούνται στα block του και είναι αρκετά καλά ορισμένη.

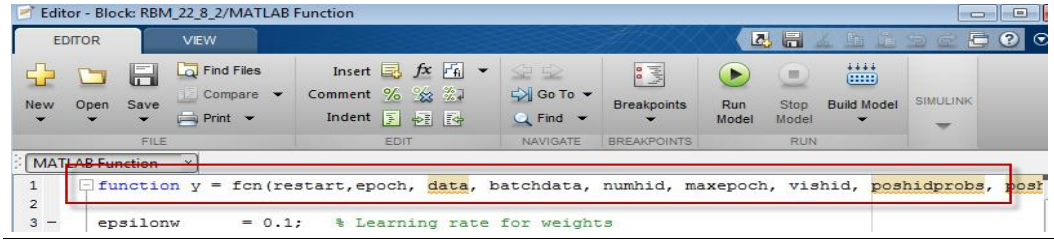
Περιγράφοντας την όλη διαδικασία ξεκινάμε με το να σώσουμε τα δεδομένα του μοντέλου μας από το workspace σε ένα .mat αρχείο και στην συνέχεια ανοίγουμε το μοντέλο μας από το Simulink και πάμε στο View -> Model Explorer. Εκεί, όπως φαίνεται παρακάτω, (1) πηγαίνουμε στο Model Workspace,(2) επιλέγουμε το data source να είναι αρχείο .mat,(3) βρίσκουμε το αρχείο που επιθυμούμε,(4) πατάμε Reinitialize from source και (5) εμφανίζονται τα δεδομένα μας ως δεδομένα του μοντέλου μας πλέον :



Εικόνα 33 Εισαγωγή αποθηκευμένων δεδομένων στο μοντέλο μας

Αφού τα δεδομένα μας περάστηκαν στο μοντέλο πρέπει να γίνει η αντιστοιχία των μεταβλητών που παίρνουν μέρος στους υπολογισμούς με αυτές που εισήχθησαν μαζί με τα δεδομένα μας.

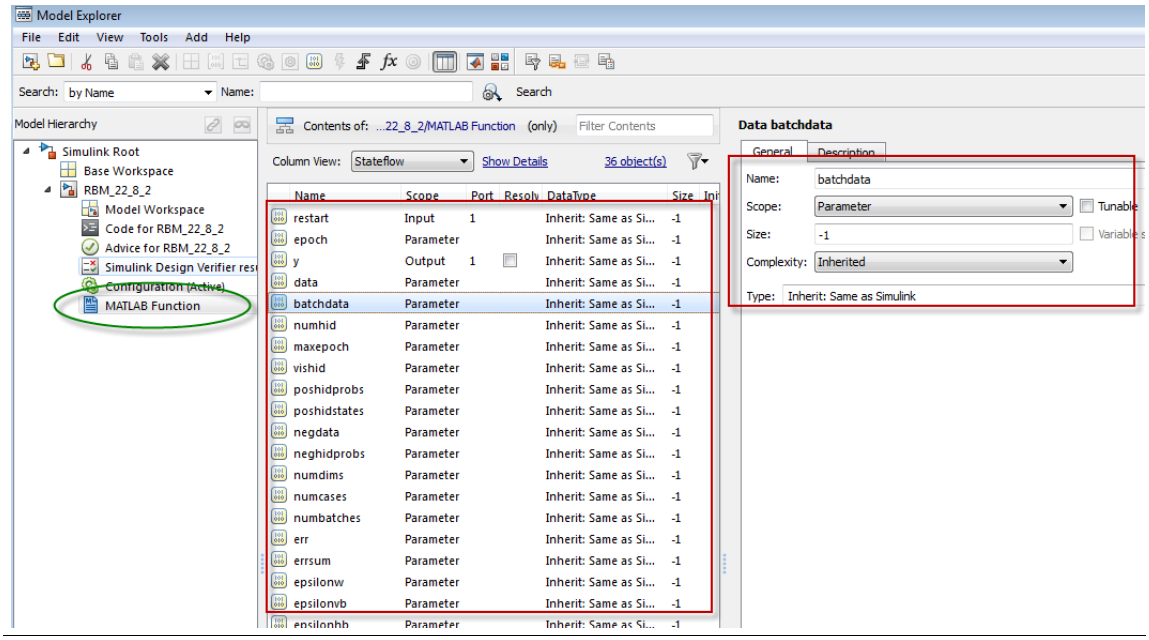
Για να χρησιμοποιηθούν μεταβλητές σε ένα Matlab Function block πρέπει να έχουν δηλωθεί στο μοντέλο μας και να έχει οριστεί ο τύπος τους (αν είναι input, output ή απλές παράμετροι). Αν ανοίξουμε τον editor του block, ώστε να δούμε τον κώδικα, διαπιστώνουμε ότι πρέπει να δηλώσουμε όποιες μεταβλητές χρησιμοποιούνται ως παραμέτρους στην κλήση της συνάρτησης



Εικόνα 34 Κλήση της συνάρτηση με παραμέτρους

Στην συγκεκριμένη γραμμή της κλήσης το y είναι η έξοδος που έχουμε αντιστοιχίσει στο σφάλμα και για τις μεταβλητές μέσα στις παρενθέσεις πρέπει να δηλώσουμε τι ακριβώς είναι η καθεμία. Έτσι πηγαίνουμε από τον Model Explorer πάνω στο συγκεκριμένο block και μπορούμε να χειριστούμε τις μεταβλητές αυτές όπως επιθυμούμε.

Όπως φαίνεται παρακάτω, ως είσοδο έχουμε δηλώσει μόνο την μεταβλητή restart και ως έξοδο μόνο την y, όπως προείπαμε. Όλες οι υπόλοιπες μεταβλητές χρησιμοποιούνται σαν απλοί παράμετροι, οι τιμές των όποιων μπορεί να είναι tunable, και επειδή έχουν ακριβώς τα ίδια ονόματα με τις μεταβλητές από τα δεδομένα μας αντιστοιχούνται από το Simulink σε αυτά. Έτσι αν κάποια μεταβλητή έχει μια τιμή από τα δεδομένα μας που πρέπει να την κρατάει δηλώνεται ως Non Tunable, όπως π.χ. batchdata όπου περιέχει τα δεδομένα των δεσμίδων κάτι που δεν θα αλλάξει κατά την διάρκεια της εκτέλεσης, ενώ για μεταβλητές που αλλάζουν τιμές όπως π.χ. τα βάρη κατά την διαδικασία εκπαίδευσης, μπορούν να είναι tunable, να αρχικοποιούνται από τα δεδομένα μας ή ακόμα και μέσα από τον κώδικα του block. Η παρακάτω εικόνα μας δείχνει το παράθυρο όπου μπορούμε να προσθαφαιρέσουμε, τροποποιήσουμε τις μεταβλητές αυτές ανάλογα με την συνάρτηση και το μοντέλο μας:

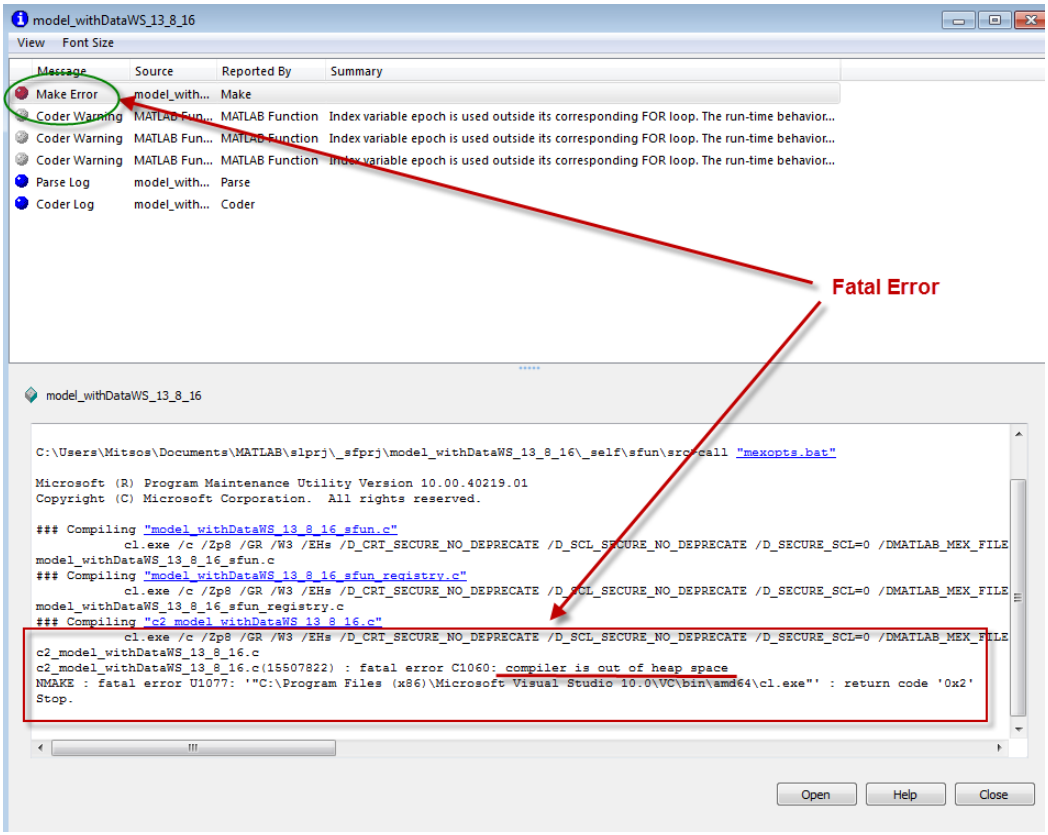


Εικόνα 35 Μεταβλητές της συνάρτησης του block

Πλέον, με το προηγούμενο βήμα, έχουμε ολοκληρώσει την διαδικασία μεταφοράς του κώδικα, των δεδομένων και των μεταβλητών ενός RBM σε επίπεδο Simulink, σε μια πρώτη απλή «αφελής» σχεδίαση.

Αν επιλέξουμε να τρέξουμε το μοντέλο μας ως έχει, το πρώτο πράγμα που θα συναντήσουμε είναι ο μεγάλος χρόνος που απαιτείται για να γίνει compile το μοντέλο μας κάτι που στο τέλος δεν γίνεται εφικτό καθώς τερματίζεται η διαδικασία. Βλέπουμε εν τέλει ότι το Simulink αδυνατεί να κάνει compile το μοντέλο μας, λόγω του ότι τα δεδομένα και οι προς εκτέλεση λειτουργίες πάνω σε αυτά, ξεπερνούν την χωρητικότητα της μνήμης του compiler. Αυτό είναι ένα γενικό πρόβλημα που συναντάται σε μοντέλα με πολλές λειτουργίες ή μεγάλο όγκο δεδομένων.

Αυτός είναι ο πρώτος μεγάλος περιορισμός που συναντούμε στην σχεδίασή μας. Αποτελεί περιορισμό by default του εργαλείου και δεν μπορεί να ξεπεραστεί παρά μόνο με μείωση του όγκου των δεδομένων ή των λειτουργιών που εκτελούνται σε αυτά. Μπορούμε να δούμε το fatal error του compiler στην παρακάτω εικόνα :



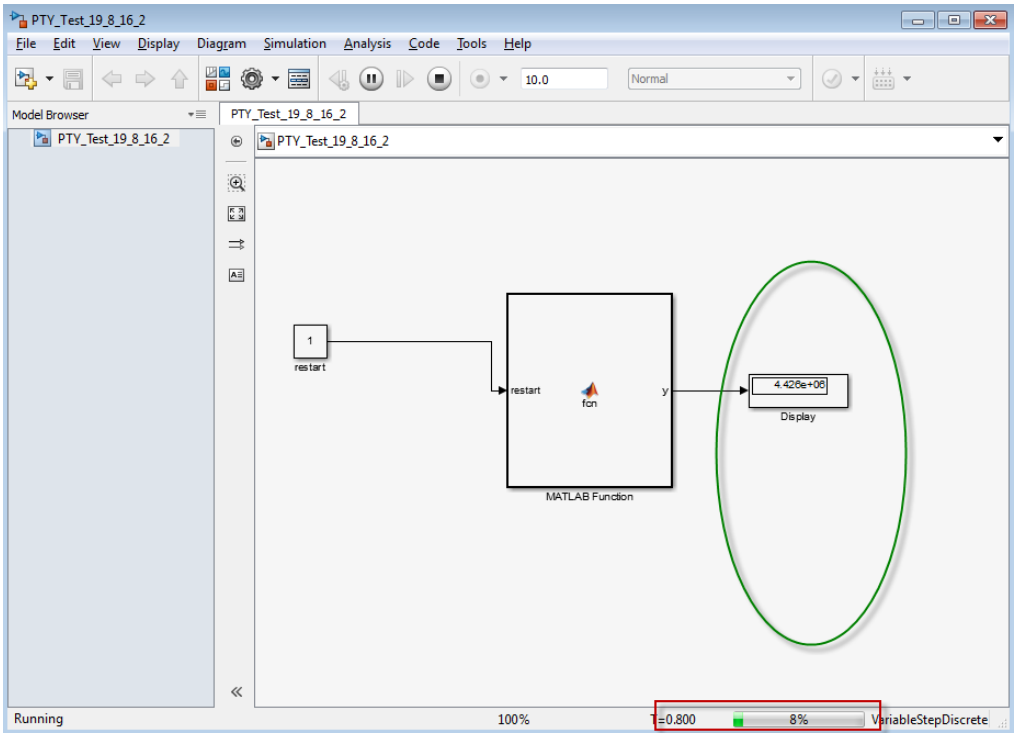
Εικόνα 36 Σφάλμα compiler, ξεπερνιέται η μέγιστη χωρητικότητα της μνήμης του compiler

Όπως προείπαμε ο μόνος τρόπος να ξεπεραστεί το πρόβλημα είναι η μείωση του όγκου των δεδομένων μας είτε η μείωση του κώδικα που εκτελείται. Σε αυτό το σημείο πρέπει να κάνουμε κάποιους συμβιβασμούς και να επιλέξουμε το πώς θα πρέπει να προχωρήσουμε με βάση του τι θέλουμε να επιτύχουμε. Επειδή μας ενδιαφέρει η λειτουργικότητα του μοντέλου περισσότερο από οτιδήποτε, επιλέξαμε να μειώσουμε τον όγκο των δεδομένων που παίρνουν μέρος στην σχεδίαση. Αυτό μπορεί να γίνει είτε αφαιρώντας δεδομένα από κάθε κλάση είτε αφαιρώντας κλάσεις μετατρέποντας το πρόβλημα σε DBN ταξινόμησης λιγότερων ψηφίων. Για να μπορούμε να κρατήσουμε την λειτουργικότητα του αρχικού κώδικα και να μπορούμε να κάνουμε σωστή εκτέλεση / παρακολούθηση της όλης διαδικασίας επιλέξαμε να μειώσουμε τα ψηφία που παίρνουν μέρος στην διαδικασία. Έτσι αντί να λειτουργήσουμε ένα DBN για ταξινόμηση 10 ψηφίων το λειτουργούμε για μικρότερο αριθμό, μειώνοντας έτσι τον όγκο των δεδομένων μας.

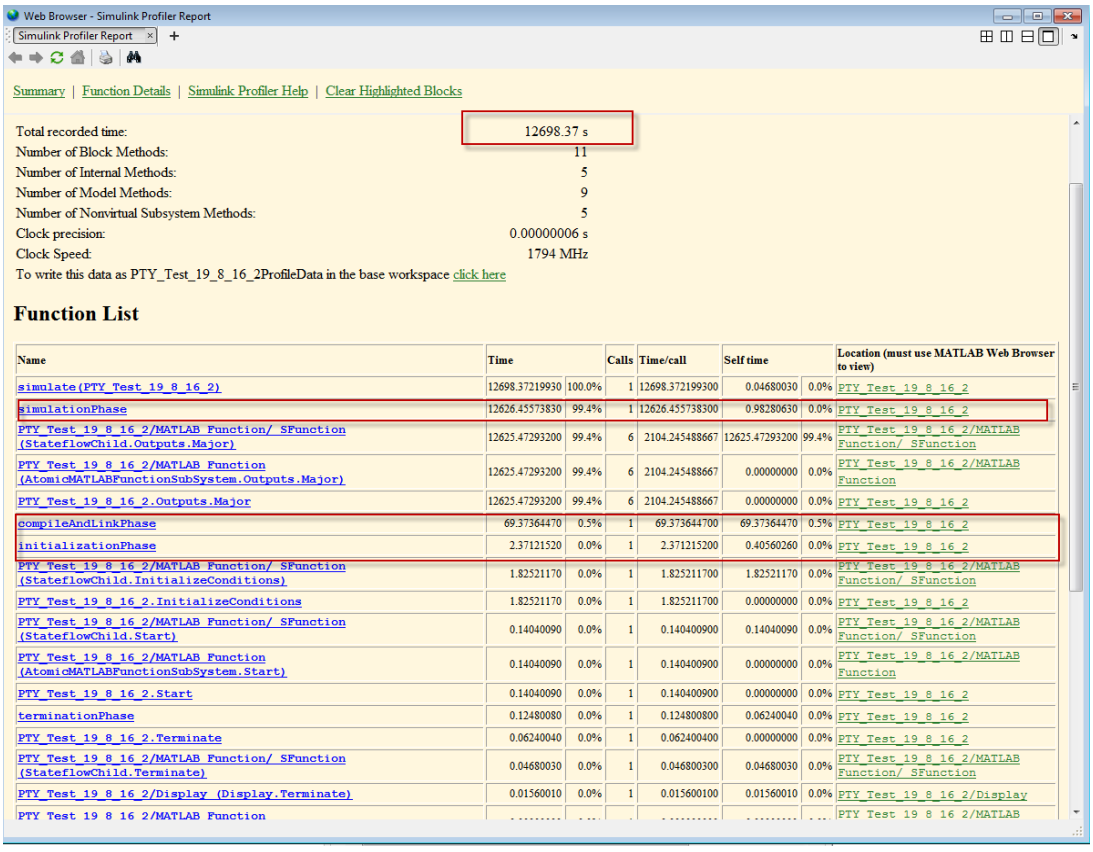
Από εδώ και στο εξής θα λειτουργήσουμε με δεδομένα που αφορούν τα 4 πρώτα ψηφία κάτι που μειώνει αρκετά τα δεδομένα μας, επισπεύδει τους υπολογισμούς και τους χρόνους μεταγλώττισης, προσομοίωσης και εκτέλεσης χωρίς να αλλάζει την βασική λειτουργία του μοντέλου μας και του κώδικα που χρησιμοποιείται.

Στην πορεία θα δούμε κάποιες επιπλέον αλλαγές που μπορούμε να κάνουμε, κυρίως στον κώδικα ώστε να βελτιώσουμε την απόδοση του και της σχεδίασης θυσιάζοντας όμως κάποια από την ακρίβεια του μοντέλου.

Με αυτήν την μείωση των δεδομένων λοιπόν μπορούμε να δοκιμάσουμε την λειτουργία της μέχρι τώρα σχεδίασης και τρέχοντας το μοντέλο μας παρατηρούμε ότι όντως το πρόβλημα της χωρητικότητας της μνήμης ξεπεράστηκε. Όμως παρά την μείωση αυτή, ο χρόνος που απαιτείται για την προσομοίωση του μοντέλου μας είναι τεράστιος. Στην παρακάτω εικόνα φαίνεται πως βρισκόμαστε στο 8% της προσομοίωσης, δηλαδή στην $4^{η}$ εποχή εκπαίδευσης. Όμως για να φτάσουμε εκεί χρειάστηκαν περίπου 211 λεπτά (3,5 ώρες).



Εικόνα 37 Λειτουργία μοντέλου για 4 ψηφία

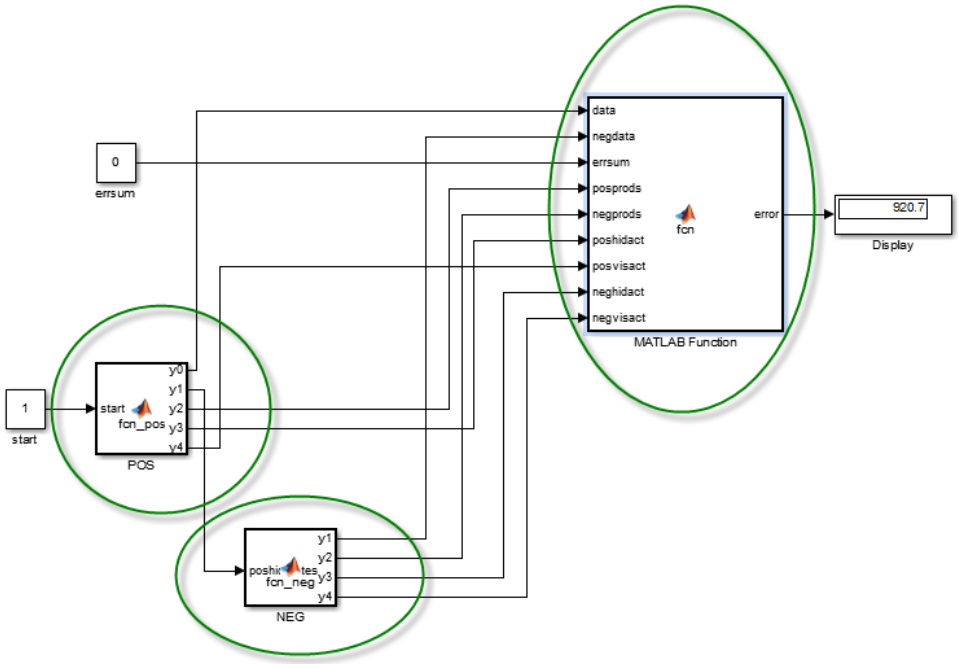


Εικόνα 38 Χρόνος που χρειάστηκε μέχρι το 8% της προσομοίωσης (12698.37 sec)

Ο απαιτούμενος χρόνος, όπως φαίνεται παραπάνω, δεν είναι αποδεκτός αλλά είναι κατανοητός καθώς μιλάμε για μια απλή αρχική σχεδίαση σε ένα μοντέλο που πραγματοποιεί, εξ' ορισμού, προσομοιώσεις, σε αντίθεση με την άμεση εκτέλεση κώδικα σε Matlab όπως είδαμε προηγουμένως (5.1).

Έχοντας ως βάση το μοντέλο που περιέχει τον κώδικα μπορούμε να προχωρήσουμε περαιτέρω και να αρχίζουμε να σπάμε την σχεδίαση σε μικρότερα κομμάτια. Αυτό θα μας βοηθήσει να «ξετυλίξουμε» τα κύρια μέρη του κώδικα και να μπορούμε στην συνέχεια να τα αντιμετωπίσουμε ξεχωριστά, μες στο μοντέλο μας. Έτσι θα μπορούμε να μεταφέρουμε όποιες λειτουργίες επιλέξουμε ξεχωριστά στο FPGA καθώς επίσης και να απλοποιηθεί το μοντέλο μας.

Μια πρώτη σωστή σχεδίαση του μοντέλου ακολουθεί παρακάτω :

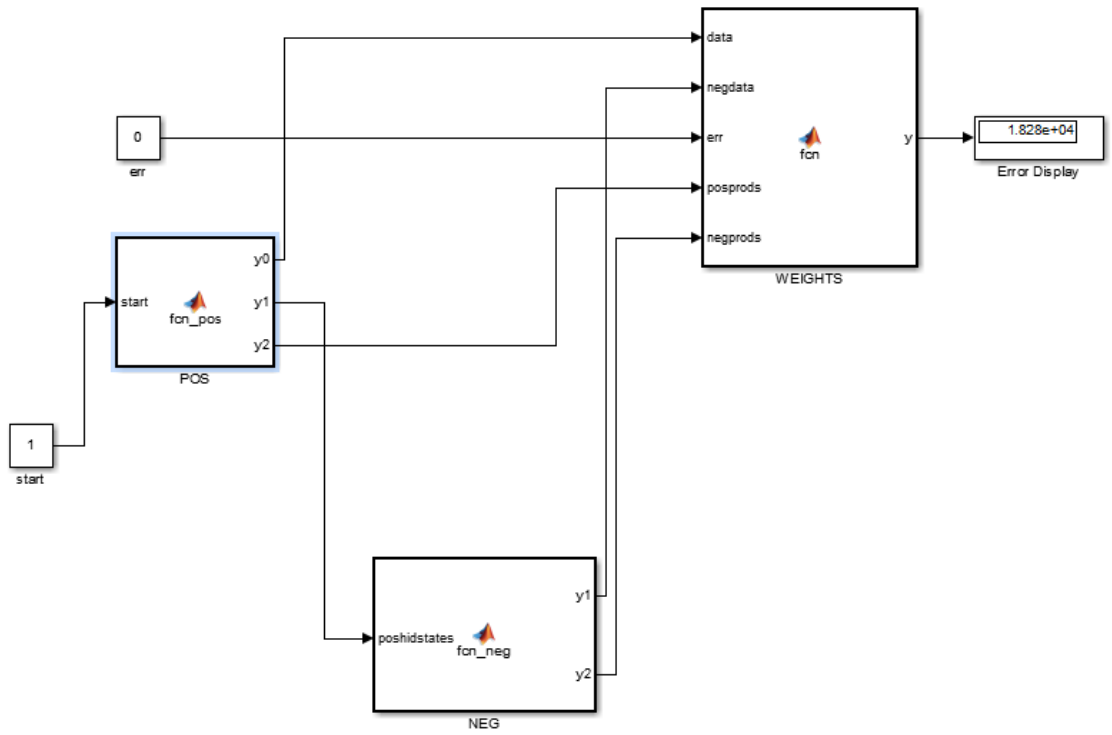


Εικόνα 39 Τμηματοποίηση λειτουργιών RBM

Όπως βλέπουμε στην προηγούμενη εικόνα (39) έχουμε πλέον μεταφέρει τον αρχικό κώδικα σε 3 επιμέρους block διαγράμματα βασισμένα στην λειτουργία του. Έτσι το πρώτο διάγραμμα είναι το POS όπου περιέχει την διαδικασία της Positive φάσης της λειτουργίας του RBM, το δεύτερο block NEG περιέχει την δεύτερη Negative φάση και το τελευταίο block που περιέχει τις ενημερώσεις των βαρών και τον biases καθώς επίσης και τον υπολογισμό του σφάλματος. Επίσης βλέπουμε τα σήματα που μεταφέρουν τις μεταβλητές από το ένα block στο άλλο και την εμφάνιση του σφάλματος στο Display block.

Προχωρώντας ένα βήμα πιο πέρα θα αφαιρέσουμε από το μοντέλο μας τις μεταβλητές, τα σήματα και τους υπολογισμούς των biases για απλούστευση της σχεδίασης και αύξησης της ταχύτητας των υπολογισμών. Αυτό, παρ' ότι μειώνει την ακρίβεια του μοντέλου, είναι εφικτό και σε αρκετές σχεδιάσεις της βιβλιογραφίας παραλείπεται καθώς μπορούμε να θεωρήσουμε ότι οι μονάδες στα δύο επίπεδα δεν έχουν προδιάθεση.

Ένα τέτοιο παράδειγμα σχεδίασης φαίνεται στην παρακάτω εικόνα :

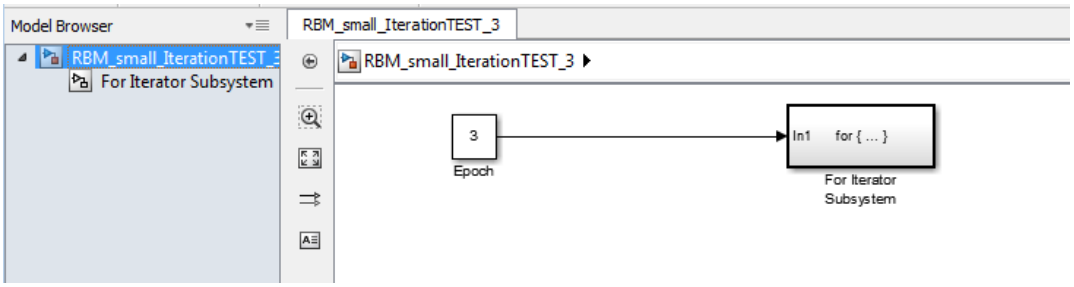


Εικόνα 40 RBM χωρίς προδιάθεση (bias)

Σε αυτό το σημείο εμφανίζεται ο δεύτερος μεγάλος περιορισμός του εργαλείου και έχει να κάνει με τις επαναληπτικές δομές και το πώς προσομοιώνονται – λειτουργούν στο Simulink. Όταν είχαμε μόνο ένα μπλοκ στην αρχική απλή σχεδίαση, οι επαναλήψεις που τρέχουν για τις εποχές και τις δεσμίδες υλοποιούνται με δύο nested for loops μες στον κώδικα. Όμως από την στιγμή που θα αρχίσουμε να προσθέτουμε block διαγράμματα και θα αρχίσει να σπάει ο κώδικας σε αυτά, δεν μπορούμε εύκολα να έχουμε αυτές τις επαναλήψεις.

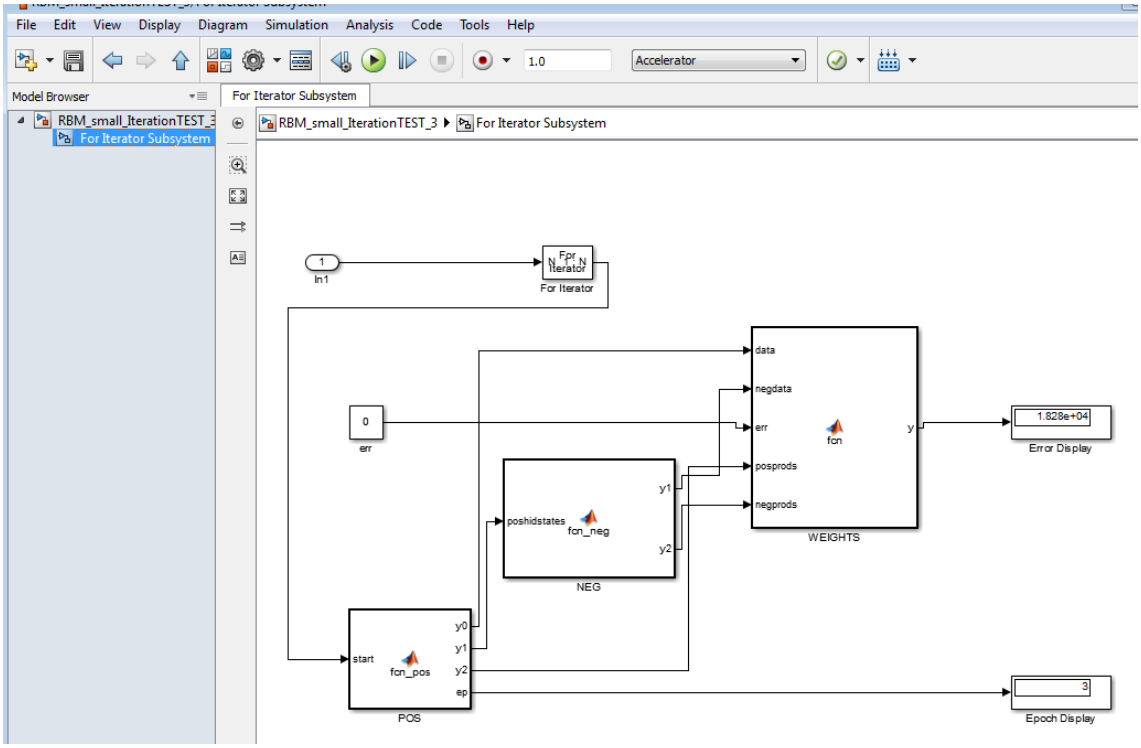
Τρεις τρόποι υπάρχουν να ξεπεραστεί αυτό το πρόβλημα. Ο πρώτος είναι να έχουμε flow control μέσα στον κώδικα σε κάθε block diagram κάτι που δεν είναι βιώσιμο, καθώς θα πρέπει να έχουμε πολλά σήματα που θα μεταδίδονται προς τα πίσω μεταξύ τον blocks και θα κάνει την εκτέλεση της προσομοίωσης πολύ αργή και δεν εγγυάται την σωστή λειτουργία του κώδικα.

Ο δεύτερος τρόπος είναι η χρησιμοποίηση των block διαγραμμάτων που παρέχονται από το εργαλείο. Αυτά είναι τα For και While Iterator Subsystems . Στην περίπτωσή μας η πιο λογική προσέγγιση είναι η χρήση του For Iterator όμως χρειαζόμαστε δύο τέτοια blocks όπου το ένα είναι nested μέσα στο άλλο. Ο τρόπος που λειτουργούν τα Iterator block προσομοιώνουν μεν τα for loops όμως δεν είναι εύχρηστα και καθυστερούν, με την σειρά τους, την προσομοίωση. Ένα παράδειγμα ενός For Iterator ακολουθεί στην παρακάτω εικόνα :



Εικόνα 41 Δήλωση For Iterator για 3 εποχές

Όταν χρησιμοποιούμε το συγκεκριμένο μπλοκ μπορούμε να ορίσουμε τις παραμέτρους που αφορούν τις επαναλήψεις που πρόκειται να γίνουν, και το τρόπο που δίνεται ο αριθμός τους. Έτσι έχουμε επιλέξει ο αριθμός τους να δίνεται εξωτερικά (constant Epoch = 3) και να τις εμφανίζουμε εσωτερικά συνδέοντας το σήμα της εξόδου, μέσα από το POS block, στο Epoch Display Block . Όπως φαίνεται παρακάτω, ανοίγοντας το block του Iterator έχουμε τα block που εκτελούνται :

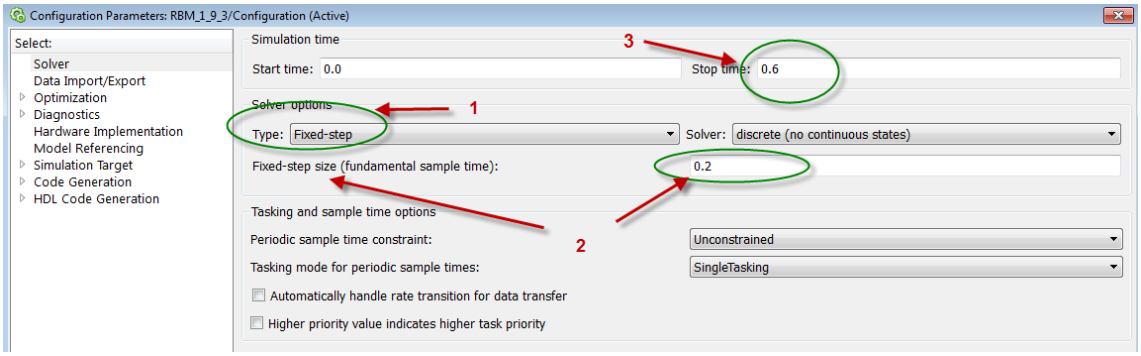


Εικόνα 42 Εκτέλεση block για 3 εποχές

Ο τρίτος τρόπος είναι η χρήση κατάλληλου Solver στο Simulink ώστε να λειτουργεί σαν επαναληπτική δομή. Ο Solver του εργαλείου αποτελεί στην ουσία τον τρόπο με τον οποίο διενεργείται η προσομοίωση (εκτέλεση) του μοντέλου μας. Χωρίς να επεκταθούμε σε περισσότερες λεπτομέρειες συνήθως χρησιμοποιείται ο default VariableStepDiscrete ode 45 ο οποίος αναλαμβάνει αυτόματα να κάνει ανάλυση της σχεδίασης και να καθορίσει τα διαστήματα που εκτελούνται τα block διαγράμματα του μοντέλου μας. Όμως αλλάζοντας τον σε FixedStep μπορούμε να καθορίσουμε τα βήματα για την εκτέλεση του μοντέλου. Υπάρχει αρκετό βάθος στην επιλογή Solver ανάλογα με το τι θέλουμε να σχεδιάσουμε [38] και αρκετές παράμετροι που πρέπει να ληφθούν υπόψιν. Έτσι δηλώνοντας ότι ολόκληρο το μοντέλο μας τρέχει μια φορά σε κάθε βήμα, μπορούμε να αντιστοιχίσουμε τα βήματα αυτά σε εποχές και να συμπεριλάβουμε έναν For Iterator για τις επαναλήψεις των δεσμίδων. Αυτός είναι και ο πιο σωστός τρόπος όμως

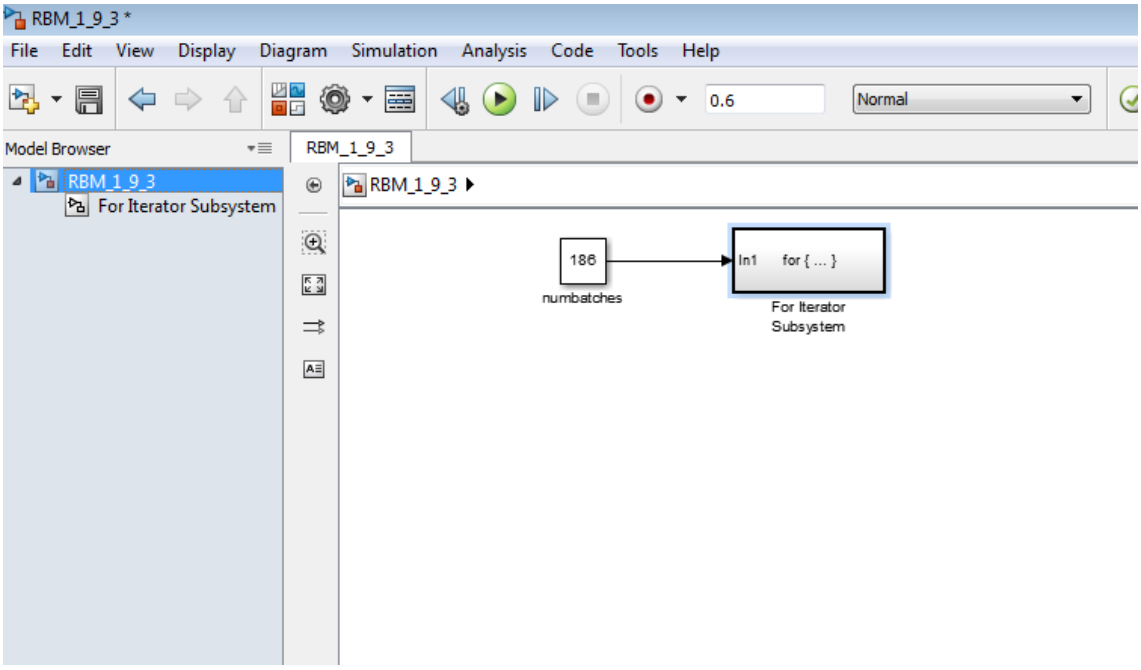
στην πράξη υπήρξαν προβλήματα που αφορούν στην σωστή λειτουργία της συγκεκριμένης δομής. Παρ' όλα αυτά δίνεται το παράδειγμα εκτέλεσης της συγκεκριμένης σχεδίασης, στις ακόλουθες εικόνες.

Όπως φαίνεται στην παρακάτω εικόνα (43), επιλέγουμε για Solver τον Fixed-step (1) και δίνουμε το Fixed-step size (2). Τέλος επιλέγουμε σε ποιο βήμα θα τελειώσει η προσομοίωση – λειτουργία του μοντέλου μας (3). Αυτό ισοδυναμεί με το τρέξιμο του μοντέλου μας για 3 εποχές, καθώς η λειτουργία ξεκινάει στο 0.0 και με βήμα 0.2 τερματίζει στο 0.6 .



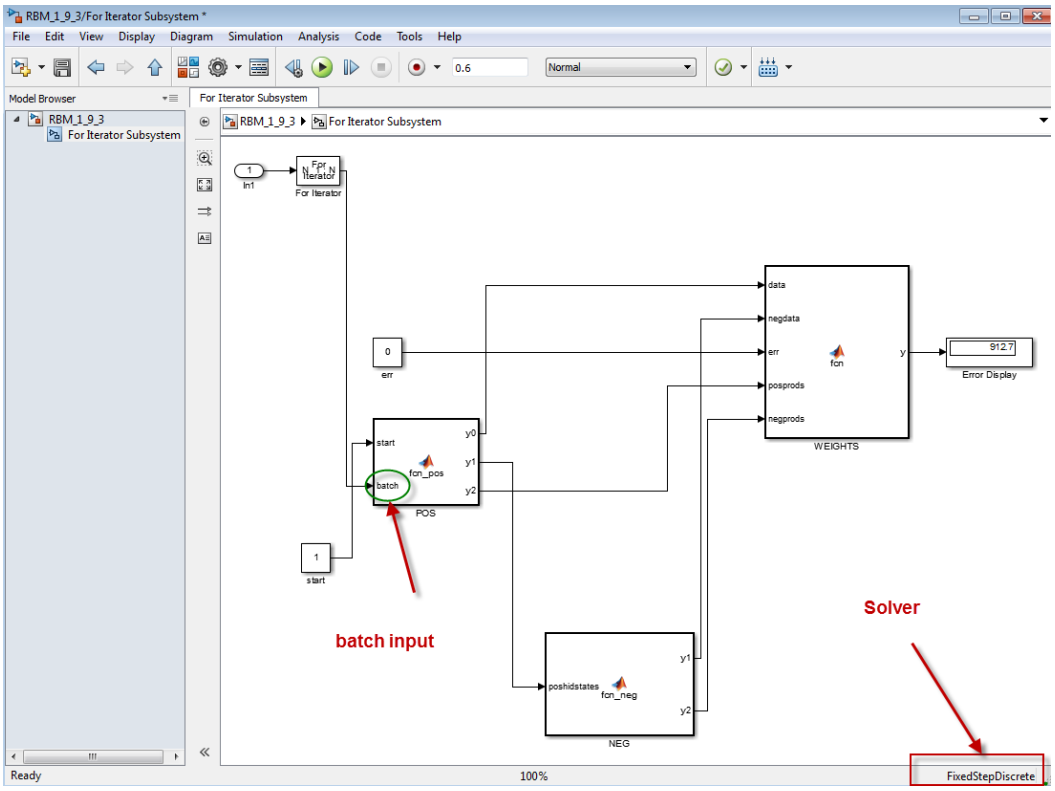
Εικόνα 43 Επιλογή Solver

Έπειτα μεταφερόμαστε στην λειτουργία των blocks και του Iterator. Όπως φαίνεται παρακάτω ο αριθμός των επαναλήψεων είναι ίσος με τον αριθμό των δεσμίδων (η πληροφορία του αριθμού των δεσμίδων βρίσκεται στα δεδομένα του μοντέλου μας στο model workspace) :



Εικόνα 44 Αριθμός επαναλήψεων του Iterator δίνεται από τον αριθμό των δεσμίδων (numbatches)

Και μέσα στο υποσύστημα του Iterator φαίνεται η εκτέλεση των block,με τον αριθμό της δεσμίδας (batch) να δίνεται ως είσοδος στο block POS όπου διαλέγει, ανάλογα με το σε ποια επανάληψη βρισκόμαστε, την αντίστοιχη δεσμίδα που παίρνει μέρος στους υπολογισμούς :



Εικόνα 45 Μοντέλο που θα εκτελεστεί για 3 εποχές και 186 δεσμίδες

Αν και η παραπάνω προσέγγιση είναι η πιο σωστή, στην πράξη δεν λειτουργεί όπως θα περιμέναμε, και παρά τις προσπάθειες μέσα από ανάγνωση εγχειριδίων και συμμετοχή σε φόρουμ δεν κατάφερα να ξεπεράσω το πρόβλημα.

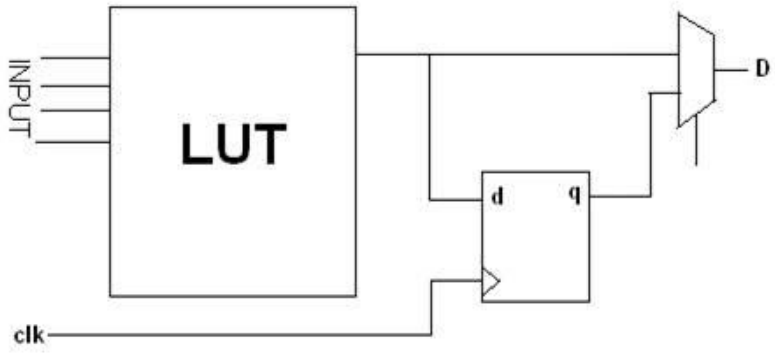
Στοχεύοντας αποκλειστικά στους υπολογισμούς που εκτελεί το RBM,στο υπόλοιπο της σχεδίασης, αφού δείξαμε παραδείγματα εκτέλεσης με For Iterator, θα επικεντρωθούμε στις λειτουργίες του χωρίς να ενσωματώνουμε τις επαναληπτικές δομές στο μοντέλο μας. Έχοντας το μοντέλο τις εικόνας 40 ως βάση θα προχωρήσουμε στην μεταφορά λειτουργιών του RBM σε επίπεδο FPGA .

5.3 FPGA Hardware in the loop μέσω Simulink

Το επόμενο βήμα είναι η μεταφορά μέρους των υπολογισμών σε επίπεδο FPGA. Αυτό γίνεται με την χρήση της τεχνικής FPGA In The Loop ή αλλιώς Hardware Software Co-Simulation. Πριν προβούμε όμως στην υλοποίηση μιας τέτοιας τεχνικής,θα δώσουμε τον ορισμό και λειτουργία των FPGAs και θα παρουσιάσουμε, εν συντομία, την πλακέτα που χρησιμοποιήσαμε στην συγκεκριμένη εργασία.

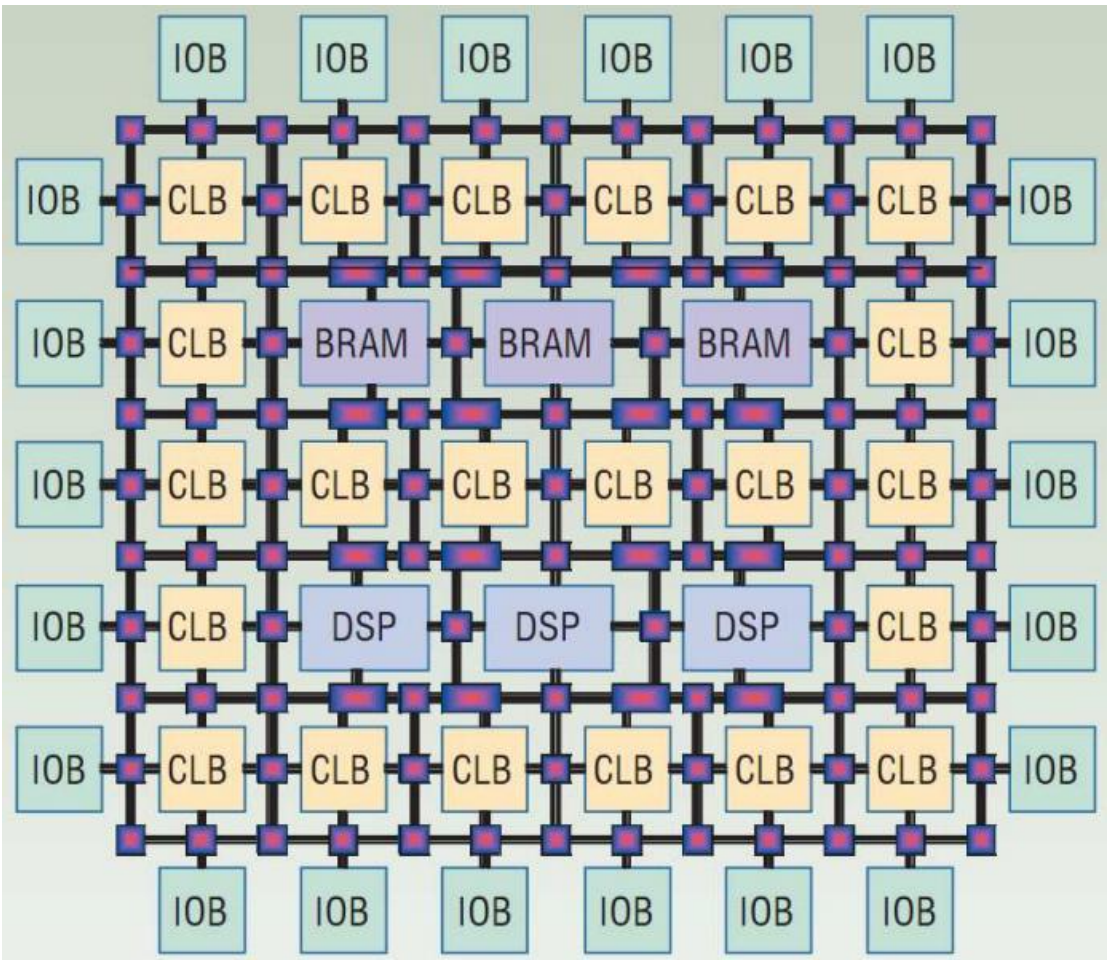
5.3.1 FPGA – Virtex 5 ML505 Platform

Τα Field Programmable Gate Arrays αποτελούν στην ουσία προγραμματιζόμενο υλικό καθώς είναι κυκλώματα ημιαγωγών που περιέχουν προγραμματιζόμενη λογική (Configurable Logic Blocks) και προγραμματιζόμενες διασυνδέσεις [39],[40]. Η κύρια δομή είναι τα logic cells που εμπεριέχονται σε ένα FPGA και απαρτίζονται από ένα Look Up Table (LUT) 4 εισόδων, ένα D-Flip Flop και έναν πολυπλέκτη 2 σε 1. Μια τυπική εικόνα ενός τέτοιου κελιού φαίνεται παρακάτω:



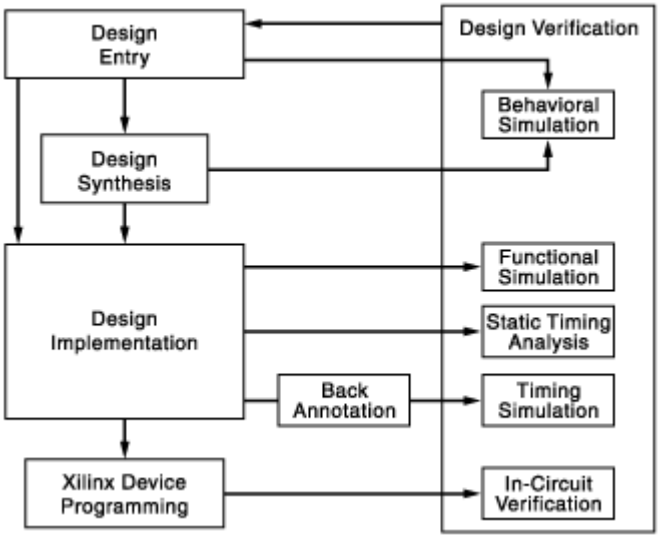
Εικόνα 46 FPGA Logic Cell

Τα CLBs αποτελούνται από logic cells, επικοινωνούν μεταξύ τους και μπορούν να συνδυαστούν και να προγραμματίζονται ώστε να υλοποιούν από απλές πύλες έως πολύπλοκες συναρτήσεις. Επιπλέον τα σύγχρονα FPGA περιέχουν επιπλέον μνήμες, πολλαπλασιαστές, αθροιστές, DSP Slices και μπορούν ακόμα να έχουν υλοποιημένους soft core επεξεργαστές. Το FPGA είναι τυπωμένο σε μια πλακέτα ,καλείται αναπτυξιακή πλατφόρμα (evaluation platform),στην οποία είναι ενσωματωμένα επιπλέον κυκλώματα και υπάρχουν αρκετές είσοδο και έξοδοι για την επικοινωνία του FPGA με περιφερειακά. Μια τυπική αρχιτεκτονική FPGA φαίνεται στην παρακάτω εικόνα :



Εικόνα 47 Τυπική Αρχιτεκτονική FPGA

Ο προγραμματισμός ενός FPGA περιλαμβάνει είτε την χρήση HDL (Hardware Description Language) που περιγράφει το πώς θα λειτουργεί το κύκλωμα που θέλουμε να δημιουργήσουμε είτε την απευθείας σχηματική σχεδίαση κυκλωμάτων. Έπειτα χρησιμοποιείται κάποιο εργαλείο αυτόματης ηλεκτρονικής σχεδίασης και παράγεται μια netlist που περιέχει όλα τα σήματα και τα κυκλώματα. Αυτή η netlist στην συνέχεια θα «προσαρμοστεί» πάνω στο FPGA μέσω της διαδικασίας τοποθέτησης και δρομολόγησης που παρέχεται από κατάλληλο software του κατασκευαστή. Ο σχεδιαστής έπειτα θα ελέγξει την σχεδίαση που έχει τοποθετηθεί πάνω στο FPGA πραγματοποιώντας χρονική ανάλυση και προσομοίωση καθώς και άλλες μεθόδους επαλήθευσης. Εφόσον η διαδικασία της σχεδίασης και επαλήθευσης ολοκληρωθεί τότε, μέσω πάλι software του κατασκευαστή, δημιουργείται ένα δυαδικό αρχείο (bitstream) το οποίο χρησιμοποιείται για τον προγραμματισμό του FPGA. Το αρχείο αυτό «κατεβαίνει» (μεταφέρεται) στο FPGA με την χρήση συνήθως JTAG σειριακής διεπαφής ή σε μια εξωτερική μνήμη πάνω στην πλακέτα που προγραμματίζει το FPGA κατά την εκκίνηση. Η γενικότερη διαδικασία φαίνεται στην ακόλουθη εικόνα :



Εικόνα 48 Διαδικασία προγραμματισμού FPGA σε μια Xilinx συσκευή

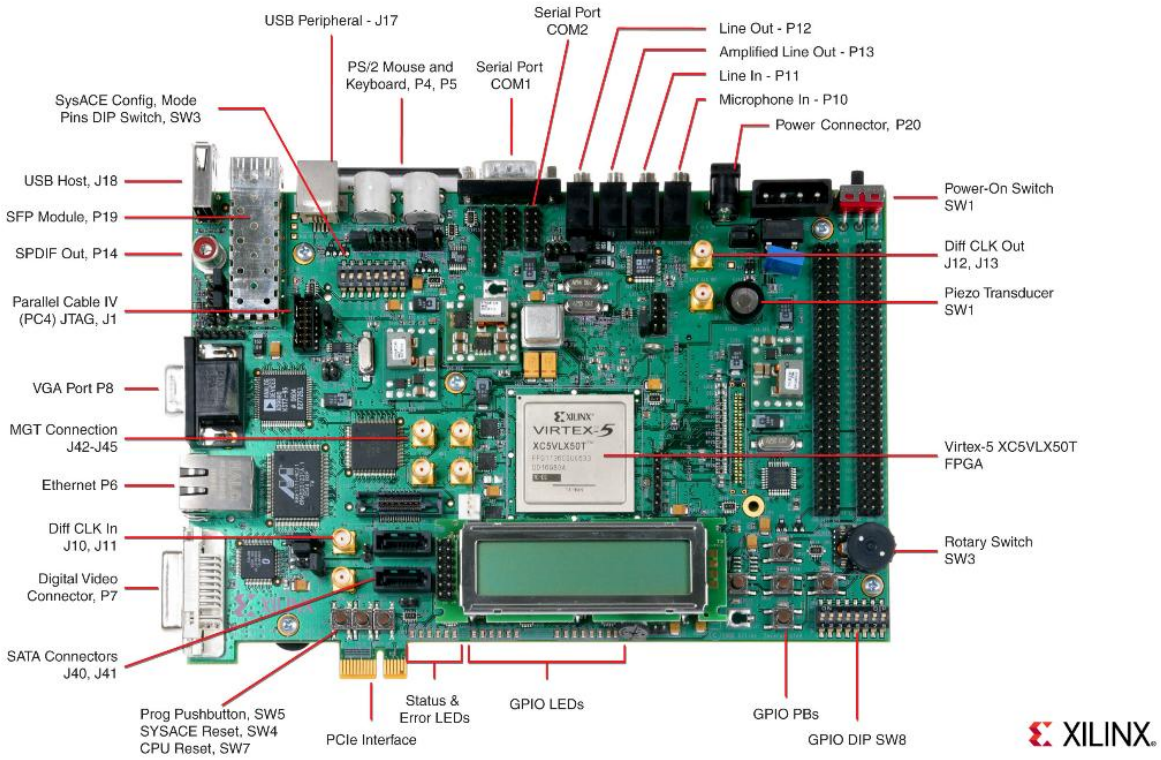
Τα κύρια πλεονεκτήματα των FPGA είναι :

- Δυνατότητα επαναπρογραμματισμού
- Παραγωγή κυκλωμάτων σε σύντομο χρονικό διάστημα
- Δυνατότητα προσομοίωσης των κυκλωμάτων
- Χαμηλό κόστος σχεδίασης και παραγωγής

Όπως είπαμε τα FPGA είναι τυπωμένα σε μια αναπτυξιακή πλακέτα που χρησιμοποιείται για προτυποποίηση. Στην περίπτωση μας η πλακέτα που θα χρησιμοποιήσουμε είναι της εταιρείας Xilinx και της οικογένειας Virtex5. Το FPGA που χρησιμοποιεί είναι το XC5VL50T και η αναπτυξιακή πλατφόρμα είναι η ML505. Ποιο συγκεκριμένα το FPGA διαθέτει CLBs με 7,200 slices και μέγιστη κατανεμημένη μνήμη 480 Kb,48 DSP48E slices και Block RAM blocks με μέγιστη χωρητικότητα 2,160 Kb [41]. Επιπλέον η ML505 πλατφόρμα, πέρα από το XC5VL50T FPGA , περιέχει 2 serial ports, Ethernet port, USB hosts, GPIOs, 256 MB DDR2 RAM και αρκετές επιπλέον συνδέσεις όπως VGA, PS2 keyboard / mouse και άλλες. Ακολουθούν εικόνες με τα specs του FPGA και της πλατφόρμας.

Device	Configurable Logic Blocks (CLBs)			DSP48E Slices(2)	Block RAM Blocks			CMTs(4)	PowerPC Processor Blocks	Endpoint Blocks for PCI Express	Ethernet MACs(5)	Max RocketIO Transceivers(6)		Total I/O Banks(8)	Max User I/O(7)
	Array (Row x Col)	Virtex-5 Slices(1)	Max Distributed RAM (Kb)		18 Kb(3)	36 Kb	Max (Kb)					GTP	GTX		
XC5VLX30	80 x 30	4,800	320	32	64	32	1,152	2	N/A	N/A	N/A	N/A	N/A	13	400
XC5VLX50	120 x 30	7,200	480	48	96	48	1,728	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX85	120 x 54	12,960	840	48	192	96	3,456	6	N/A	N/A	N/A	N/A	N/A	17	560
XC5VLX110	160 x 54	17,280	1,120	64	256	128	4,608	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX155	160 x 76	24,320	1,640	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX220	160 x 108	34,560	2,280	128	384	192	6,912	6	N/A	N/A	N/A	N/A	N/A	23	800
XC5VLX330	240 x 108	51,840	3,420	192	576	288	10,368	6	N/A	N/A	N/A	N/A	N/A	33	1,200
XC5VLX20T	60 x 26	3,120	210	24	52	26	936	1	N/A	1	2	4	N/A	7	172
XC5VLX30T	80 x 30	4,800	320	32	72	36	1,296	2	N/A	1	4	8	N/A	12	360
XC5VLX50T	120 x 30	7,200	480	48	120	60	2,160	6	N/A	1	4	12	N/A	15	480

Εικόνα 49 XC5VLX50T FPGA specs



Εικόνα 50 ML505 Front

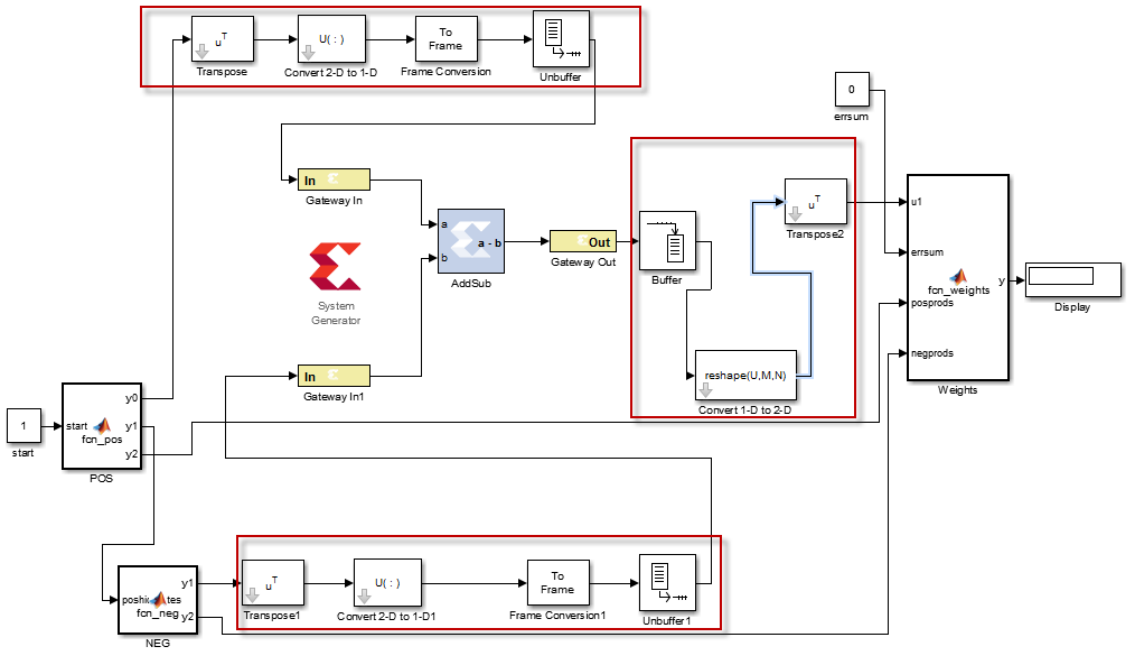
Εικόνα 51 ML505 Back

5.3.2 Simulink – FPGA In the loop

Όπως προαναφέραμε η FPGA In the Loop τεχνική μας επιτρέπει την σύνδεση ενός FPGA στον υπολογιστή και την επικοινωνία του με περιβάλλον software. Χρησιμοποιείται ως ενδιάμεσο στάδιο, μεταξύ των εισόδων και των εξόδων του software, με σκοπό την επιτάχυνση των υπολογισμών. Υπολογισμοί που στο software χρειάζονται μεγάλους χρόνους εκτέλεσης μπορούν να μεταφερθούν σε επίπεδο hardware όπου θα εκτελεστούν ταχύτερα. Στο παράρτημα Β μπορούμε να δούμε την διαδικασία που ακολουθείται για μια FPGA In the Loop σχεδίαση σε περιβάλλον Matlab Simulink.

Έχοντας λοιπόν στο μυαλό μας την λειτουργία των FPGAs, το RBM της εικόνας 40, καθώς επίσης και τις υλοποιήσεις που είδαμε στο κεφάλαιο 4 μπορούμε να ξεκινήσουμε να σχεδιάζουμε την μετάβαση στο FPGA. Όπως έχει διαμορφωθεί η σχεδίαση έως τώρα υπάρχουν δύο τρόποι που θα μπορούσαμε να προχωρήσουμε.

Ο πρώτος τρόπος είναι μια On the fly λογική γνωρίζοντας ότι οι υπολογισμοί που γίνονται σε επίπεδο hardware (FPGA) είναι ταχύτεροι από αυτούς σε software πάνω από γενικού σκοπού επεξεργαστές. Έτσι κάνοντας πάλι μικρά βήματα θα μπορούσαμε, για αρχή, να μεταφέρουμε κομμάτι των υπολογισμών σχεδιάζοντας FPGA blocks που τρέχουν μέρους των κομματιών του κώδικα από τα blocks της σχεδίασης μας. Η μεταφορά ενός απλού υπολογισμού ,διαφορά των positive και negative data που μας δίνει στην ουσία το σφάλμα, φαίνεται στην παρακάτω εικόνα:



Εικόνα 52 Μεταφορά υπολογισμού του σφάλματος σε FPGA

Στην προηγούμενη εικόνα βλέπουμε τα blocks του RBM ,όπως τα είχαμε σχεδιάσει στην εικόνα 40, με την διαφορά ότι πλέον ο υπολογισμός του σφάλματος (data –negdata) μεταφέρεται στο FPGA. Οι συγκεκριμένοι πίνακες (data, negdata) είναι 784x500 και βλέπουμε ότι για να λειτουργήσουν σωστά οι υπολογισμοί On the Fly στο κύκλωμα χρειάζονται κάποιοι ενδιάμεσοι buffers και μετατροπή των πινάκων σε διανύσματα ώστε να μεταφερθούν σωστά στο FPGA. Αυτό προκύπτει από την φύση του προβλήματος και των δεδομένων μας καθώς αποτελούνται σε πολύ μεγάλο ποσοστό από πίνακες και μάλιστα αρκετά μεγάλους σε μέγεθος. Αφού έχουν μετασχηματιστεί, εισάγονται στο FPGA με την χρήση των Gateway In blocks, που έχουν κίτρινο χρώμα. Έπειτα ο υπολογισμός που εκτελείται, φαίνεται στο Add/Sub block το οποίο εμφανίζεται με μπλε χρώμα. Τέλος μεταφέρεται ο υπολογισμός πίσω στο μοντέλο με την χρήση του Gateway Out block και γίνεται ξανά μετασχηματισμός σε πίνακα πριν χρησιμοποιηθεί εκ νέου από το μοντέλο. Τα μπλοκ που σχετίζονται με το FPGA έχουν πάντα κάποιο χρώμα ενώ τα μπλοκ που λειτουργούν σε επίπεδο Simulink είναι πάντα ασπρόμαυρα.

Στην παρακάτω εικόνα φαίνεται ο κώδικας που εκτελείται στο κάθε block και ποια σήματα έχουν μεταφερθεί στο FPGA :

POS

```
function [y0,y1,y2] = fcn_pos(start,data,batchdata,batch,poshidprobs,vishid,numcases,batch
%%%%%%%%% START POSITIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  data = batchdata(:,:,batch); %Visible units V
  poshidprobs = 1./(1 + exp(-data*vishid ))%Updating hidden states (ph)
  batchposhidprobs(:,:,batch)=poshidprobs;%Setting hidden layer states for every batch
  posprods    = data' * poshidprobs;% V*H Data (positive statistics)

%%%%%%%%% END OF POSITIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  poshidstates = poshidprobs > rand(numcases,numhid);% Firing up neurons in hidden layer
y0 = data;
y1 = poshidstates;
y2 = posprods;
start = 0;
```

NEG

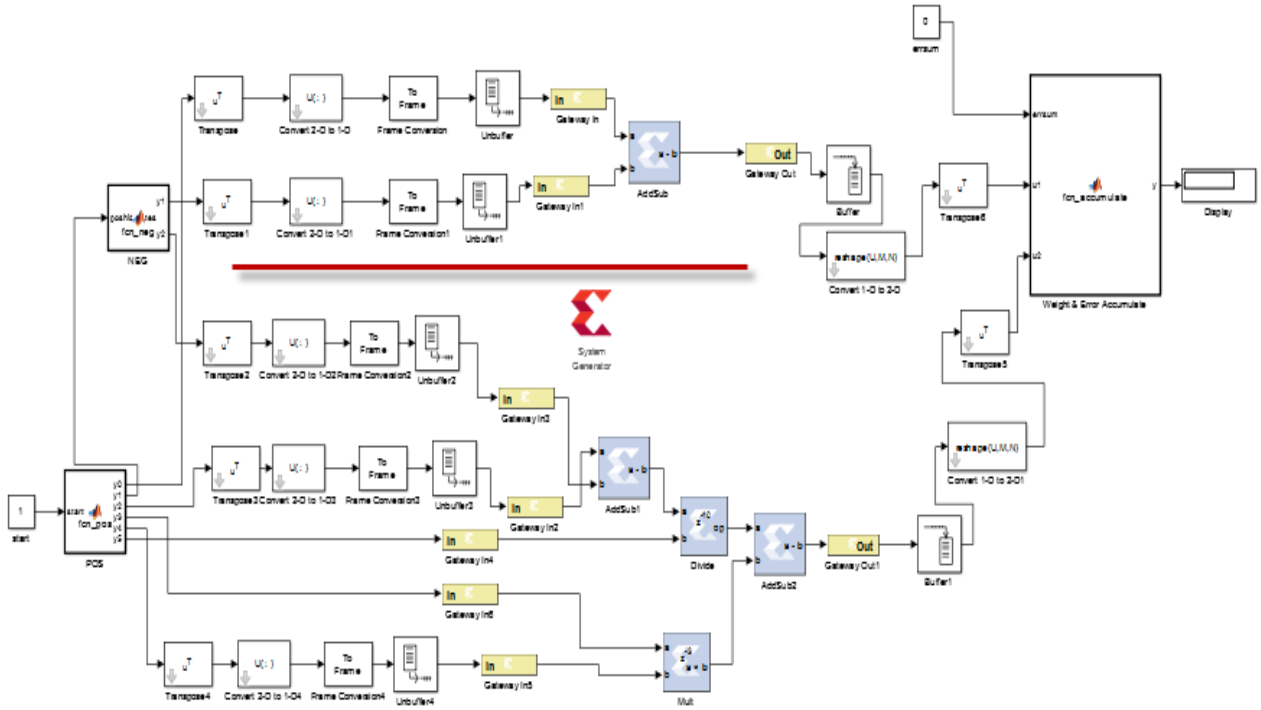
```
function [y1,y2] = fcn_neg(poshidstates,negdata,vishid,numcases,neghidprobs,negprods)

%%%%%%%%% START NEGATIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
  negdata = 1./(1 + exp(-poshidstates*vishid'))% Updating Visible states (pv)
  neghidprobs = 1./(1 + exp(-negdata*vishid))% Updating hdden states reconstruction
  negprods  = negdata'*neghidprobs;% V*H Model

%%%%%%%%% END OF NEGATIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
y1 = negdata;
y2 = negprods;
```

Weights

```
function y = fcn_weights( u1, numcases, err,vishid, errsum, epsilonw, weightcost, posp

err= sum(sum( (u1).^2 ));
errsum = err + errsum;

%%%%%%%%% UPDATE WEIGHTS AND BIASES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    vishidinc = momentum*vishidinc + ...
                epsilonw*( (posprods-negprods)/numcases - weightcost*vishid);
%%%%%%%%%%%%%%%% END OF UPDATES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

y = errsum;
```

Εικόνα 53 Κώδικας των block και μεταφορά σφάλματος data - negdata (u1) στο FPGA

Παρατηρούμε ότι κάθε είσοδος στο FPGA, εφόσον αποτελεί πίνακα, χρειάζεται μετατροπή και unbuffer πριν εισαχθεί στο FPGA και ξανά buffer και μετατροπή πριν γυρίσει πίσω στο μοντέλο. Μπορούμε επιπλέον να μεταφέρουμε και άλλες λειτουργίες από τα υπόλοιπα blocks καταλήγοντας σταδιακά στην μεταφορά ολόκληρων των υπολογισμών στο FPGA. Στην εικόνα που ακολουθεί φαίνεται η επιπλέον μεταφορά υπολογισμών στο FPGA και συγκεκριμένα ο υπολογισμός της ανανέωσης των βαρών :



Εικόνα 54 Μεταφορά υπολογισμού ανανέωσης των βαρών και σφάλματος

Σε αυτό το σημείο, ας ρίξουμε μια ματιά στον κώδικα των block ώστε να δούμε πιο υπολογισμοί έχουν μεταφερθεί σε επίπεδο FPGA και ποιοι εξακολουθούν να βρίσκονται σε επίπεδο Simulink και γιατί.

Ξεκινώντας με το πρώτο block POS έχουμε :

POS

```
1   function [y0, y1,y2,y3,y4,y5]  = fcn_pos(start,data,batchdata,batch,poshidprobs,vishid,numcases,b
2   %%%%%%%%% START POSITIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3     data = batchdata(:,:,batch);
4     poshidprobs = 1./(1 + exp(-data*vishid));
5     batchposhidprobs(:,:,batch)=poshidprobs;
6     posprods    = data' * poshidprobs;
7
8   %%%%%%%%% END OF POSITIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9    poshidstates = poshidprobs > rand(numcases,numhid);
10  y0 = data;
11  y1 = poshidstates;
12  y2 = posprods;
13  y3 = weightcost;
14  y4 = vishid;
15  y5 = numcases;
16  start = 0;
```

Εικόνα 55 POS Block κώδικας

Η **γραμμή 3** στον προηγούμενο κώδικα θέτει στα data μας τα δεδομένα του συγκεκριμένου batch , δηλαδή μας δίνει το διάνυσμα του ορατού επιπέδου μας V για το συγκεκριμένο batch. Αυτή είναι πράξη αρχικοποίησης που δεν έχει νόημα να μεταφερθεί στο FPGA ,καθώς δεν αποτελεί υπολογισμό. Η **γραμμή 4** υπολογίζει το κρυφό επίπεδο H ,με βάση το ορατό(data) και τα βάρη (vishid). Είναι μια διαίρεση με παρονομαστή που έχει εκθετικούς όρους και δεν είναι εύκολη η μεταφορά της στο FPGA. Σε τέτοιες περιπτώσεις συνήθως σχεδιάζεται custom hardware για την αποδοτικότερη λειτουργία τέτοιων υπολογισμών. Η **γραμμή 5** είναι και αυτή

μια ανάθεση του κρυφού επιπέδου σε ολόκληρο το batch. Η **γραμμή 6** μας δίνει τα στατιστικά για τα data μας ($V^T H$ ή $\langle v_i h_j \rangle_{data}$ Contrastive Divergence). Εκτελεί αναστροφή και πολλαπλασιασμό πινάκων και χρειάζεται και αυτή με την σειρά της αρκετούς πόρους και custom σχεδίαση για να μεταφερθεί σε FPGA επίπεδο. Η **γραμμή 9** στην ουσία ενεργοποιεί ανάλογα τους νευρώνες στο κρυφό επίπεδο αφού κάνει την σύγκριση με έναν generator τυχαίων αριθμών. Και αυτή η πράξη δεν μεταφέρθηκε σε FPGA καθώς χρειάζεται και αυτή custom κώδικα που δημιουργεί τον αντίστοιχο random generator και κάνει την σύγκριση. Οι **γραμμές 10-15** μεταφέρουν τα αντίστοιχα σήματα στις εξόδους του block ώστε να χρησιμοποιηθούν στην συνέχεια είτε από το μοντέλο είτε από το FPGA.

Προχωρώντας στο επόμενο NEG block έχουμε :

NEG

```
1   function [y1,y2]  = fcn_neg(poshidstates,negdata,vishid,numcases,neg
2
3   %%%%%%%%%% START NEGATIVE PHASE  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
4     negdata = 1./(1 + exp(-poshidstates*vishid'));
5     neghidprobs = 1./(1 + exp(-negdata*vishid));
6     negprods  = negdata'*neghidprobs;
7
8   %%%%%%%%%% END OF NEGATIVE PHASE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
9   y1 = negdata;
10  y2 = negprods;
11
```

Εικόνα 56 NEG block κώδικας

Εδώ πλέον έχουμε με την διαδικασία της φάσης reconstruct της Contrastive Divergence μεθόδου, όπου έχοντας την κατάσταση των κόμβων στο κρυφό επίπεδο (poshidstates έξοδος από το προηγούμενο POS block) και τα βάρη, δημιουργούμε το ορατό επίπεδο (**γραμμή 4**) και μετά ξανά για αυτό το επίπεδο υπολογίζουμε το κρυφό (**γραμμή 5**). Έπειτα πλέον συλλέγουμε τα στατιστικά για το μοντέλο στην **γραμμή 6** ($V^T H$ model ή $\langle v_i h_j \rangle_{model}$ Contrastive Divergence). Έπειτα δίνουμε ως έξοδο του block τα negdata, τα οποία θα χρησιμοποιηθούν στον υπολογισμό του σφάλματος, και τα negprods που θα χρησιμοποιηθούν στην ανανέωση των βαρών. Στην εικόνα 54 το κομμάτι των υπολογισμών που έχει τοποθετηθεί πάνω από το block του System Generator εκτελεί τον υπολογισμό του σφάλματος στο FPGA ενώ όλοι οι υπολογισμοί που εκτελούνται από κάτω από το block υπολογίζουν την ανανέωση των βαρών.

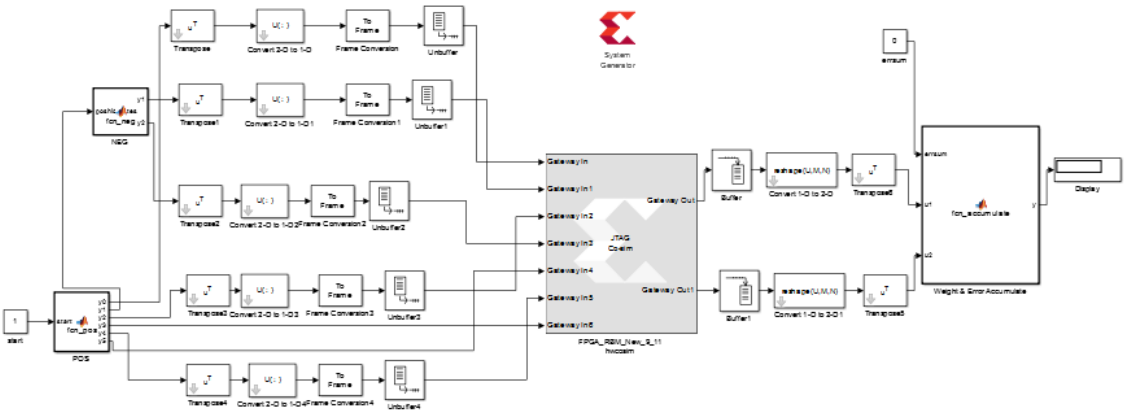
Έτσι βλέπουμε ότι στο τελευταίο block στο Simulink έχουν μείνει οι συσσωρευτικοί υπολογισμοί όπως φαίνεται και στον ακόλουθο κώδικα :

Weight & Error Accumul...

```
1   function y = fcn_accumulate(err, errsum,epsilonw,vishidinc,vishid,momentum,u1,u2)
2     err= sum(sum( (u1).^2 ));
3     errsum = err + errsum;
4
5
6   %%%%%%%%%% UPDATE WEIGHTS AND BIASES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
7     vishidinc = momentum*vishidinc + epsilonw*u2;
8     vishid = u2 + vishidinc;
9   %%%%%%%%%%%%%%%% END OF UPDATES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
10
11  y = errsum;
```

Εικόνα 57 Weight & Error Accumulate block κώδικας

Το τελικό μας μοντέλο θα έχει την ακόλουθη μορφή όπου το γκρι μπλοκ θα εκτελεστεί στο FPGA :



Εικόνα 58 RBM FPGA In the Loop

Το ζητούμενο των παραπάνω σχεδιάσεων είναι στην ουσία να μην αποθηκεύονται τα δεδομένα στην πλακέτα παρά μόνο να γίνονται streaming μέσα από αυτήν για να εκτελεστούν οι υπολογισμοί που απαιτούνται. Στην περίπτωσή μας τα δεδομένα μεταφέρονται σειριακά στην συσκευή μέσω JTAG διασύνδεσης. Αυτό σημαίνει πολύ μικρές ταχύτητες μεταφοράς . Υπάρχει η δυνατότητα, ανάλογα με την πλακέτα και την έκδοση των εργαλείων, η μεταφορά να γίνεται μέσω Ethernet κάτι που την κάνει πολύ πιο γρήγορη. Η πλακέτα που χρησιμοποιήθηκε κατά την διάρκεια της εργασίας όμως επιτρέπει την σύνδεση μόνο με χρήση JTAG. Γενικότερα στην FPGA In the Loop τεχνική ο πιο περιοριστικός παράγοντας, από άποψη επιδόσεων, είναι επικοινωνία μεταξύ του software και του hardware. Στην δίκη μας περίπτωση, λόγω της φύσης του αλγορίθμου και των πολλών επαναληπτικών λειτουργιών, αυτή η συνεχής επικοινωνία του Simulink με το FPGA κάνει την όλη υλοποίηση πολύ αργή. Επιπλέον οι ρυθμίσεις ώστε να λειτουργούν σωστά οι ενδιάμεσοι buffers και οι αντίστοιχές μετατροπές περιορίζουν σχετικά την γενικότητα την σχεδίασης, καθώς για κάθε διαφορετική υλοποίηση θα πρέπει να αλλάζουν τα μεγέθη από τον χρήστη, σε επίπεδο Simulink.

Ο δεύτερος τρόπος είναι να γίνεται μαζικά η μεταφορά μεγάλων κομματιών δεδομένων στο FPGA και να αποθηκεύονται τοπικά σε αυτό. Αυτό σημαίνει ότι πρέπει να υπάρχουν οι κατάλληλες δομές,buffers και RAMs στο FPGA. Μια τέτοια υλοποίηση δεν δημιουργήθηκε στα πλαίσια της παρούσης εργασίας κυρίως λόγω έλλειψης χρόνου και επιπλέον δεν θα μείωνε την πολυπλοκότητα της σχεδίασης. Αυτό γιατί ενώ μπορούμε να μεταφέρουμε με Burst τρόπο δεδομένα στην πλακέτα θα πρέπει να σχεδιάσουμε επιπλέον δομές συγκέντρωσης των δεδομένων στο Matlab – Simulink, να σχεδιάσουμε την μεταφορά με burst τρόπο (μέσω Ethernet) στο FPGA και να προβούμε σε δημιουργία – τοποθέτηση επιπλέον μνημών μες στο FPGA. Σε αυτήν την περίπτωση έχουμε να αντιμετωπίσουμε την ακαταλληλότητα του εργαλείου για την σωστή σχεδίαση και τοποθέτηση πάνω στο FPGA. Όχι με την έννοια τόσο ότι το εργαλείο δεν θα λειτουργήσει σωστά όσο του ότι μια εξειδικευμένη σχεδίαση σε HDL και με χρήση κατάλληλων εργαλείων του κατασκευαστή για την τοποθέτηση αυτή, θα λειτουργούσε πολύ αποδοτικότερα. Μια τοποθέτηση μέσω Simulink τέτοιων δομών δεν κάνει καλή εκμετάλλευση του FPGA και προϋποθέτει hardware abundance (αφθονία σε πόρους hardware). Σε αυτό το σημείο εισέρχεται και η έννοια της προσαρμοστικότητας / παραμετροποιησιμότητας του μοντέλου καθώς επίσης και του επιτρεπτού χώρου που μπορούμε να έχουμε για να αποθηκεύσουμε τα δεδομένα μας, κάτι που είναι ο πιο σημαντικός ανασταλτικός παράγοντας για μια τέτοια σχεδίαση.

Όσο αφορά στην προσαρμοστικότητα, για να σχεδιάσουμε τέτοιες δομές στο FPGA θα πρέπει να δώσουμε τα ακριβή μεγέθη αυτών κάτι που μπορεί να μην ισχύει για την γενική περίπτωση και θα λειτουργήσει μόνο για την συγκεκριμένη υλοποίηση. Αν θέλουμε να

ξαναχρησιμοποιήσουμε το hardware που μόλις σχεδιάσαμε για διαφορετικό αριθμό δεδομένων (π.χ. αριθμό νευρώνων στα δύο επίπεδα) θα πρέπει να επανασχεδιάσουμε τις μνήμες στο FPGA κάτι που αποτελεί μειονέκτημα του όλου εγχειρήματος.

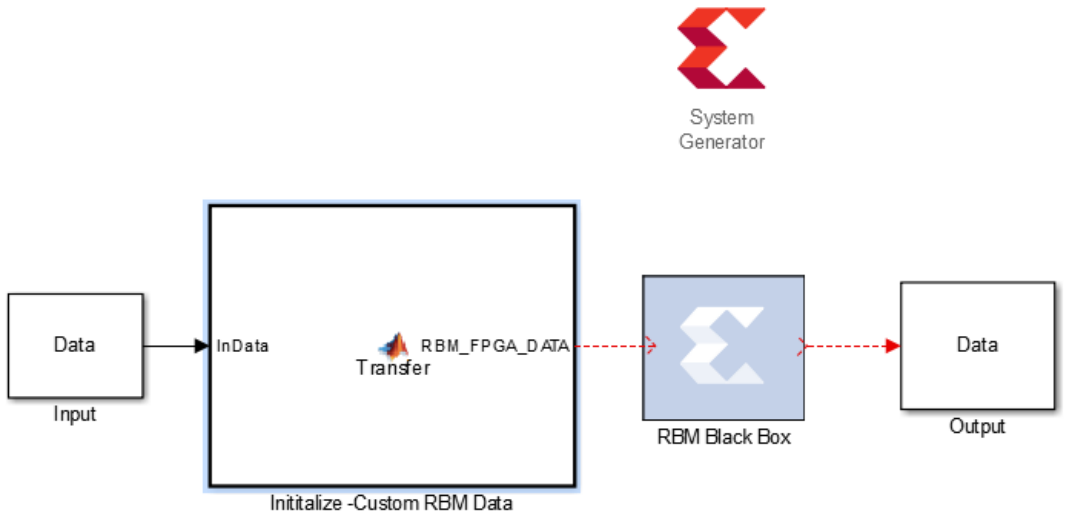
Έπειτα, σε δεύτερο επίπεδο, οι μνήμες αυτές μπορούν να έχουν συγκεκριμένο μέγεθος, καθώς υπάρχει περιορισμός από άποψη των διαθέσιμων πόρων που υπάρχουν στο εκάστοτε FPGA. Έτσι μπορεί αυτή η τοπική αποθήκευση να μην καλύπτεται από το FPGA μας και ακόμα και αν καλύπτεται από αυτό δεν είναι σίγουρο ότι θα καλύπτεται και από άλλα FPGA. Υπάρχουν πλήθος πινάκων με μεγάλο μέγεθος που λαμβάνουν μέρος στους υπολογισμούς. Σε συνδυασμό με τον αριθμό των επαναλήψεων και το γεγονός ότι τα δεδομένα μας, λόγω του ότι προέρχονται από «πραγματικό» και όχι ιδεατό data set, αποτελούνται από real value τιμές των 32 bit καταλαβαίνουμε ότι οι ανάγκες αποθήκευσης στο FPGA είναι σχεδόν απαγορευτικές.

5.3.3 Επιπλέον βήματα / πρόταση βελτίωσης

Βλέπουμε ότι παρά το γεγονός ότι μπορούμε να έχουμε λειτουργικές σχεδιάσεις RBM με χρήση FPGA In the loop δεν είναι ιδανικές, η κάθε μία λόγω διαφορετικών συνθηκών. Αυτό βέβαια δεν σημαίνει ότι είναι εξολοκλήρου αποτρεπτικές. Μελετώντας τα προτερήματα και μειονεκτήματα των διαφόρων προσπαθειών που έχουν γίνει για μεταφορά ενός RBM σε FPGA, μπορούμε να καταλήξουμε σε μια χρυσή τομή μεταξύ απόδοσης, ταχύτητας, προσαρμοστικότητας και γενίκευσης. Το επόμενο βήμα είναι ο συνδυασμός των προσεγγίσεων που επικρατούν για την μεταφορά RBM σε επίπεδο hardware και πιο συγκεκριμένα σε FPGA.
Αυτή η άποψη στηρίζεται σε δύο κύριες παραμέτρους / παρατηρήσεις.
1) Όσο αφορά στο εργαλείο Matlab Simulink υπάρχει συνεχείς εξέλιξη και ενσωματώνονται ολοένα περισσότερες λειτουργίες και δυνατότητες. Επιπλέον αυξάνεται η υπολογιστική ισχύς και η χωρητικότητα μνήμης, σε επίπεδο Ηλεκτρονικών Υπολογιστών, κάτι που κάνει την προσομοίωση τέτοιων συστημάτων ακόμη πιο γρήγορη. Ενδεικτικά οι χρόνοι τρεξίματος, προσομοίωσης και compilation ήταν αρκετά μεγάλοι (από ώρες μέχρι μέρες για μέρος του dataset) καθ' όλη την διάρκεια της εργασίας, όμως όλοι οι υπολογισμοί πραγματοποιήθηκαν σε Intel Core duo επεξεργαστή με 4GB μνήμης. Ο συνδυασμός λοιπόν εργαλείου και υπολογιστικών πόρων μπορεί να λειτουργήσει θετικά στην προσαρμοστικότητα του μοντέλου καθώς μπορεί να είναι βάση για την μεταφορά και εξατομίκευση του RBM που θέλουμε να μεταφέρουμε σε ένα FPGA.

2) Οι τεχνολογίες FPGA βελτιώνονται και αυτές με την σειρά τους, παράγοντας ταχύτερες και μεγαλύτερες, σε χώρο / πόρους, συσκευές με επιπλέον έτοιμες υλοποιήσεις DSP blocks. Το Simulink επιτρέπει την εισαγωγή Custom Made FPGA blocks εφόσον αυτά έχουν σχεδιαστεί και επαληθευτεί σωστά. Έτσι μπορούμε να σχεδιάσουμε, με χρήση HDL, custom hardware εξειδικευμένο στους γενικούς υπολογισμούς ενός RBM, όπως είδαμε και στην ενότητα 4 αλλά με μια πιο γενική προσέγγιση. Αυτό θα αυξήσει την αποδοτικότητα σε επίπεδο υπολογισμών και θα κάνει την εκτέλεση αρκετά πιο γρήγορη. Εκτός αυτού θα είναι κομμάτι «κλειστό» που δεν θα χρειάζεται παραμετροποίηση από τον χρήστη, παρά μόνο τοποθέτησή του σε μια Simulink σχεδίαση.

Εν κατακλείδι μια υλοποίηση σε Simulink με χρήση custom RBM blocks και FPGA In the loop Co Simulation θα μπορεί να εκμεταλλευτεί τα πλεονέκτημα και των δύο προσεγγίσεων, και να εξαλείψει σε μεγάλο ποσοστό τα πολλά μειονεκτήματα που παρουσιάζονται ξεχωριστά για κάθε προσέγγιση. Μια τέτοια σχεδίαση, σε ένα πρώτο αφηρημένο επίπεδο, θα έχει την ακόλουθη εικόνα.



Εικόνα 59 Αφηρημένο επίπεδο για μια βέλτιστη σχεδίαση

Τέλος αξίζει να αναφέρουμε κάποιες επιπλέον παρατηρήσεις. Το Simulink όπως προείπαμε είναι ενσωματωμένο στο Matlab. Οι παράμετροι των block του μπορούν να προγραμματιστούν μέσω μερικών γραμμών κώδικα σε Matlab, που ορίζουν τιμές στα διάφορα χαρακτηριστικά (attributes) αυτών. Αυτό σημαίνει ότι σε όλα τα επίπεδα, από το πλήθος των κόμβων μέχρι το μέγεθος των buffer, μπορούν να είναι εύκολα παραμετροποιήσιμα.

Έχοντας το Custom RBM Data block ως βάση καθένας μπορεί να το παραμετροποιήσει σε σχέση με το RBM που θέλει να υλοποιήσει χωρίς να αλλάξει η σχεδίαση στο Simulink. Το RBM Black Box (FPGA) εκτελεί σταθερά On the Fly υπολογισμούς που εγγυούνται μέγιστη ταχύτητα. Το μοντέλο της προηγούμενης εικόνας με την συνεισφορά ενός Simulink και ενός Hardware expert μπορεί να λειτουργήσει ως το επόμενο βήμα στην προσπάθεια για αξιόλογες υλοποιήσεις DBN.

Υπάρχουν αρκετοί παράγοντες, όπως είδαμε, που κάνουν τα RBM μη εφαρμόσιμα στην άμεση επίλυση προβλημάτων και μια τέτοια σχεδίαση μπορεί να ανοίξει τον δρόμο για την χρησιμοποίησή τους σε real life εμπορικές και βιομηχανικές εφαρμογές.

6. Συμπεράσματα

Ανακεφαλαιώνοντας, είδαμε ότι το πεδίο της Μηχανική Μάθησης προσφέρει αλγορίθμους που επιλύουν πολύπλοκα προβλήματα που αφορούν σε μεγάλους όγκους δεδομένων . Οι αλγόριθμοι αυτοί αποτελούν στην ουσία Τεχνητά Νευρωνικά Δίκτυα ικανά για πρόβλεψη, ταξινόμηση και data mining διαδικασίες. Ένα τέτοιο state of the art Τεχνητό Νευρωνικό Δίκτυο είναι τα Restricted Boltzmann Machines. Είδαμε πως λειτουργούν ως γενετικά στοχαστικά μοντέλα ικανά για πρόβλεψη και ταξινόμηση. Αναλύσαμε τον τρόπο εκπαίδευσής τους και τους κανόνες που τα διέπουν και εστιάσαμε στην μεγάλη γκάμα εφαρμογών που βρίσκουν. Το πιο σημαντικό χαρακτηριστικό τους είναι ότι αποτελούν την βάση για ακόμη πιο δυνατά δίκτυα, τα Deep Belief Networks. Δείξαμε την σπουδαιότητα τους με την χρήση ενός real life παραδείγματος και το πώς μπορούν να αναγνωρίσουν ψηφία γραμμένα με το χέρι χρησιμοποιώντας ένα πολύ μεγάλο data set αταξινόμητων δεδομένων.

Δεδομένου του μεγάλου χρόνου που απαιτείται για την εκπαίδευση και λειτουργία τέτοιων δικτύων, υπάρχει άμεση ανάγκη για υλοποιήσεις που θα μπορούν να αποδώσουν ταχύτερα. Έτσι έχουμε οδηγηθεί σε προσπάθειες μεταφοράς RBM σε επίπεδο hardware. Πιο συγκεκριμένα αναλύσαμε τις κύριες προσπάθειες που έχουν γίνει για την μεταφορά RBM υπολογισμών σε FPGAs. Οι προσπάθειες αυτές, αν και αυξάνουν την αποδοτικότητα των υπολογισμών, συνοδεύονται από πλήθος περιορισμών και παραμένουν πρακτικά μη εφαρμόσιμες. Αυτό λόγω προβλημάτων που έχουν να κάνουν με την παραμετροποίηση αυτών και με τον όγκο δεδομένων που μπορούν διαχειριστούν.

Για να εστιάσουμε στην επίλυση αυτών των προβλημάτων προτείναμε την υλοποίηση μιας FPGA In the Loop σχεδίασης ικανής για μέγιστη παραμετροποίηση και χειρισμού όγκου δεδομένων. Ξεκινήσαμε από την ανάλυση του Matlab κώδικα ενός DBN και περάσαμε στην λεπτομερή σχεδίαση ενός RBM σε επίπεδο Simulink. Είδαμε αναλυτικά όλα τα βήματα για να χτίσουμε μια τέτοια σχεδίαση ξεκινώντας από τον κώδικα και την μεταφορά του σε blocks Simulink, τον τρόπο εισαγωγής των δεδομένων και τους περιορισμούς του εργαλείου. Δείξαμε πως πρέπει να ληφθούν αρκετές παράμετροι υπόψιν και πως υπάρχουν ακόμα προβλήματα που πρέπει να λυθούν. Με βάση αυτό το μοντέλο προχωρήσαμε και αναλύσαμε την διαδικασία υλοποίησης της FPGA In the loop σχεδίασης, και παρότι καταφέραμε να τρέξουμε μια τέτοια σχεδίαση είδαμε πως και αυτή έχει με την σειρά της κάποιους επιπλέον περιορισμούς και δυσκολίες. Παρ' όλα αυτά πραγματοποιώντας μια λεπτομερή ανάλυση στους ανασταλτικούς παράγοντες των προηγούμενων σχεδιάσεων σε FPGA και κατανοώντας τους λόγους δυσλειτουργίας της δικής μας προσέγγισης μπορούμε να εξάγουμε πολύ χρήσιμα συμπεράσματα.

Αυτά τα συμπεράσματα μας οδήγησαν στην συνειδητοποίηση ότι η πιο σωστή προσέγγιση είναι ένας συμβιβασμός μεταξύ ταχύτητας και παραμετροποιησιμότητας. Έτσι ένα ιδανικό μοντέλο θα έχει ως βάση το Simulink για την εισαγωγή, προεπεξεργασία και παραμετροποίηση των δεδομένων του εκάστοτε RBM που θέλουμε να υλοποιήσουμε, και ως πυρήνα του Custom Made Hardware σχεδιασμένο σε FPGA που εκτελεί τις γενικές λειτουργίες ενός RBM On the fly χωρίς να χρειάζεται η αποθήκευση δεδομένων σε αυτό και άρα μπορεί να χρησιμοποιηθεί σε οποιαδήποτε σχεδίαση ανεξαρτήτου δεδομένων.

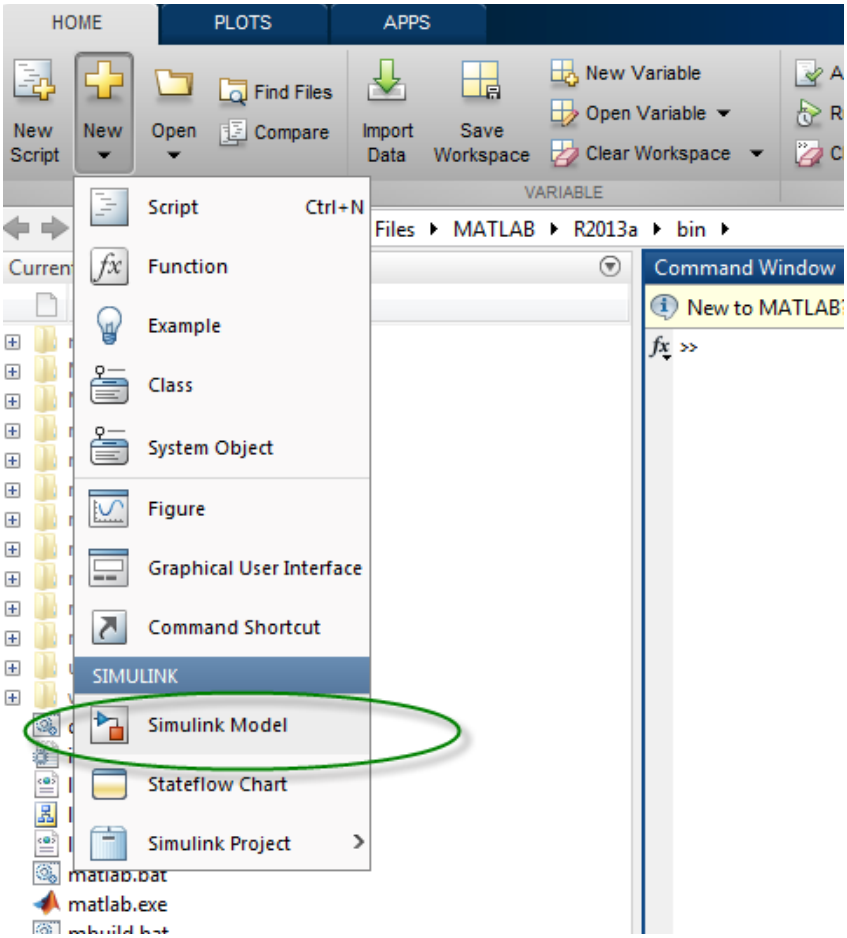
Καταλαβαίνουμε αμέσως πως μια τέτοια υλοποίηση θα έχει πλήθος εφαρμογών σε διαφορετικά πεδία. Αρκεί να δίνονται οι κατάλληλες παράμετροι που περιγράφουν το πρόβλημα προς επίλυση και το μοντέλο μας θα αναλαμβάνει την μεταφορά αυτών και τον υπολογισμό τους σε πολύ καλούς χρόνους με ελάχιστη προσπάθεια από πλευράς χρήστη.

Μια τέτοια σχεδίαση θα μπορεί να αναδειχθεί σε ένα σημαντικό εργαλείο στα χέρια των ερευνητών και να ωθήσει τον κλάδο της Μηχανικής Μάθηση ακόμη πιο πέρα. Με συνεχής βελτιώσεις και με την εισαγωγή νέων τεχνολογιών και υλοποιήσεων ένα τέτοιο εργαλείο θα μπορούσε να κάνει την χρήση DBN αποδοτική για εμπορικές ή και βιομηχανικές εφαρμογές.

ΠΑΡΑΡΤΗΜΑ Α – Παράδειγμα Simulink

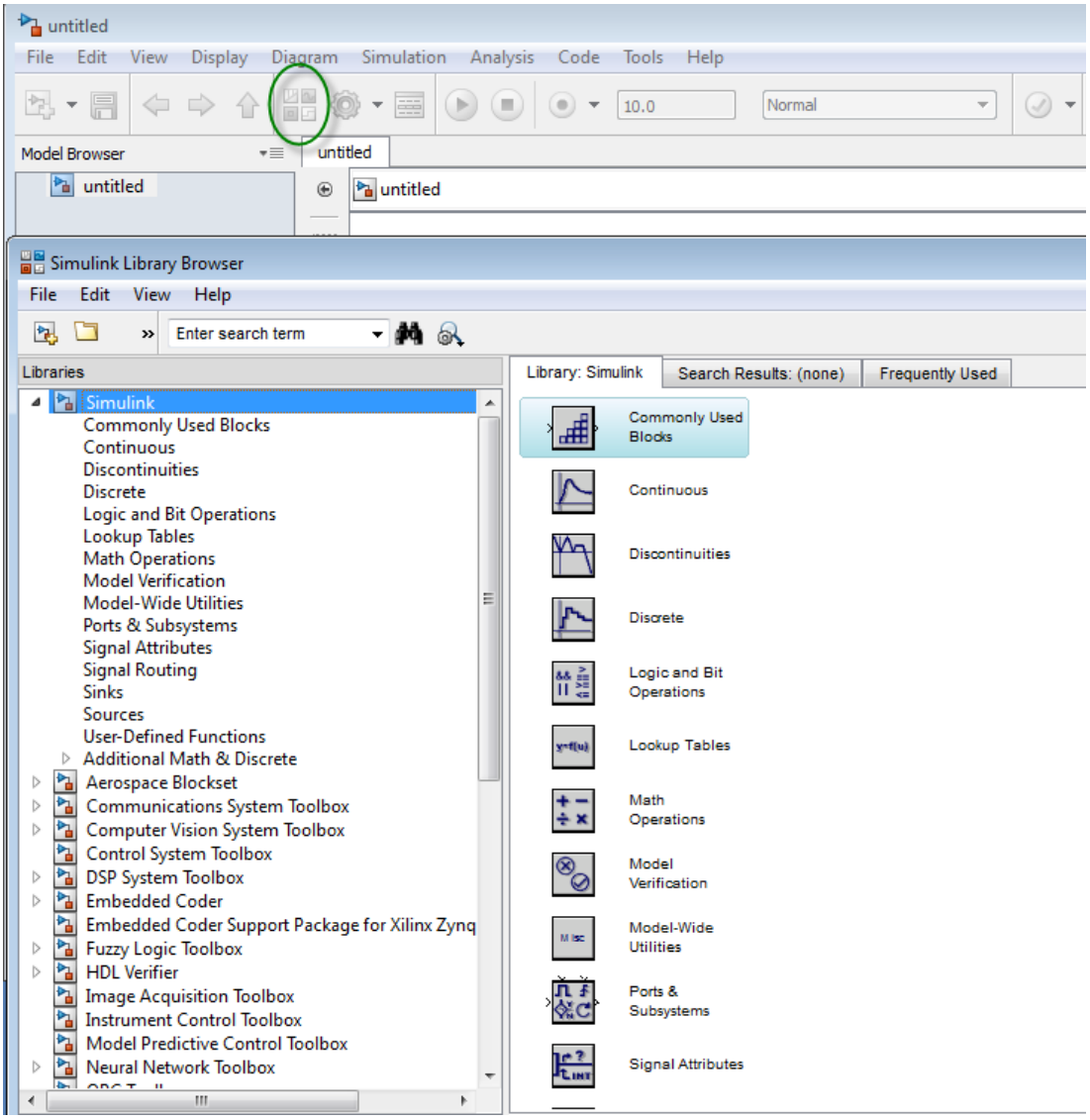
Μια τυπική σχεδίαση ενός μοντέλου στο Simulink περιέχει μια γεννήτρια σημάτων, τα μπλοκ επεξεργασίας αυτών και την παραγόμενη έξοδο. Μια τέτοια διαδικασία περιγράφεται στις παρακάτω εικόνες.

Ξεκινώντας μπορούμε είτε να καλέσουμε το Simulink από το command window του Matlab είτε να το επιλέξουμε από το μενού όπως φαίνεται παρακάτω:



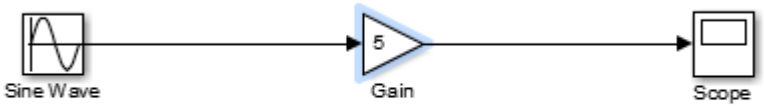
Εικόνα 60 Δημιουργία νέου μοντέλου Simulink

Πατώντας πάνω στο Simulink Library Browser μας δίνεται πληθώρα επιλογών, και ανάλογα με το τι σύστημα θέλουμε να μοντελοποιήσουμε, μπορούμε να επιλέξουμε τα blocks που θα χρησιμοποιήσουμε :



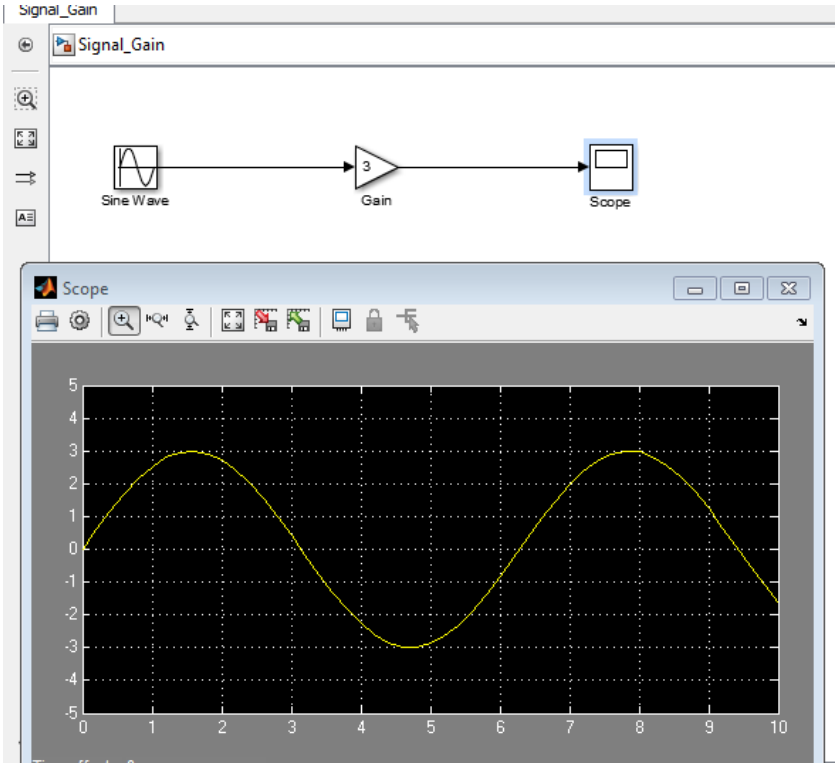
Εικόνα 61 Simulink Library Browser

Στην συνέχεια σχεδιάζουμε ένα απλό σύστημα με μια γεννήτρια σημάτων ένα μπλοκ ενίσχυσης σήματος (gain) και την προβολή της εξόδου με την χρήση του Scope block :

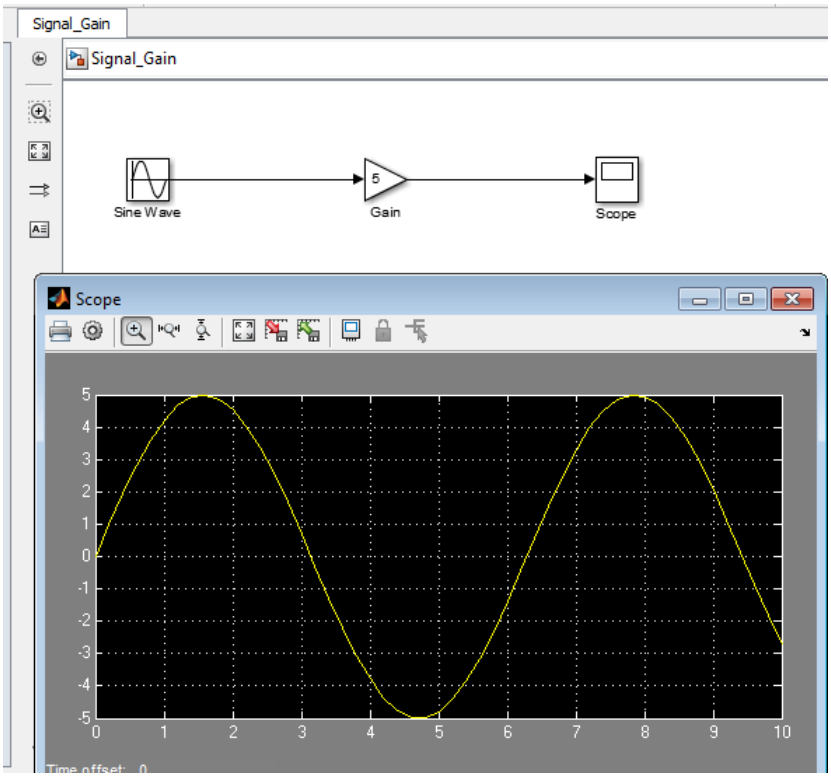


Εικόνα 62 Απλή ενίσχυση σήματος σε Simulink

Τρέχοντας το μοντέλο για Gain = 3, 5 και ανοίγοντας το Scope block βλέπουμε τα παραγόμενα αποτελέσματα :



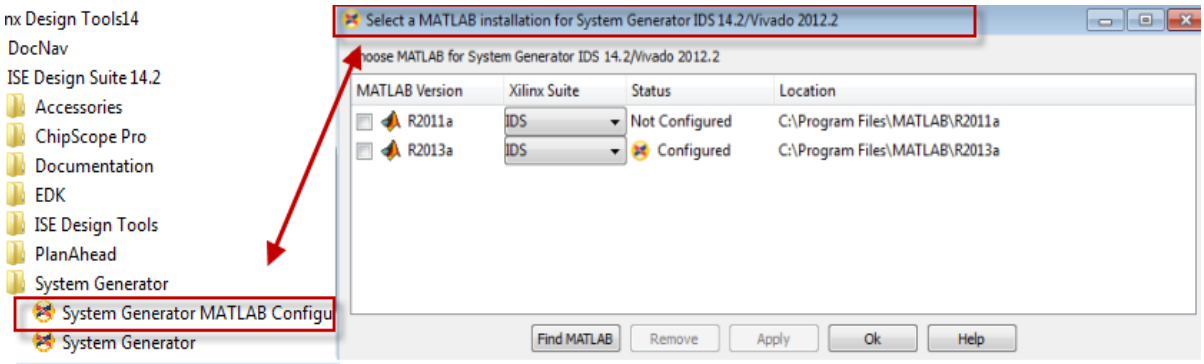
Εικόνα 63 Gain = 3



Εικόνα 64 Gain =5 για το ίδιο σήμα

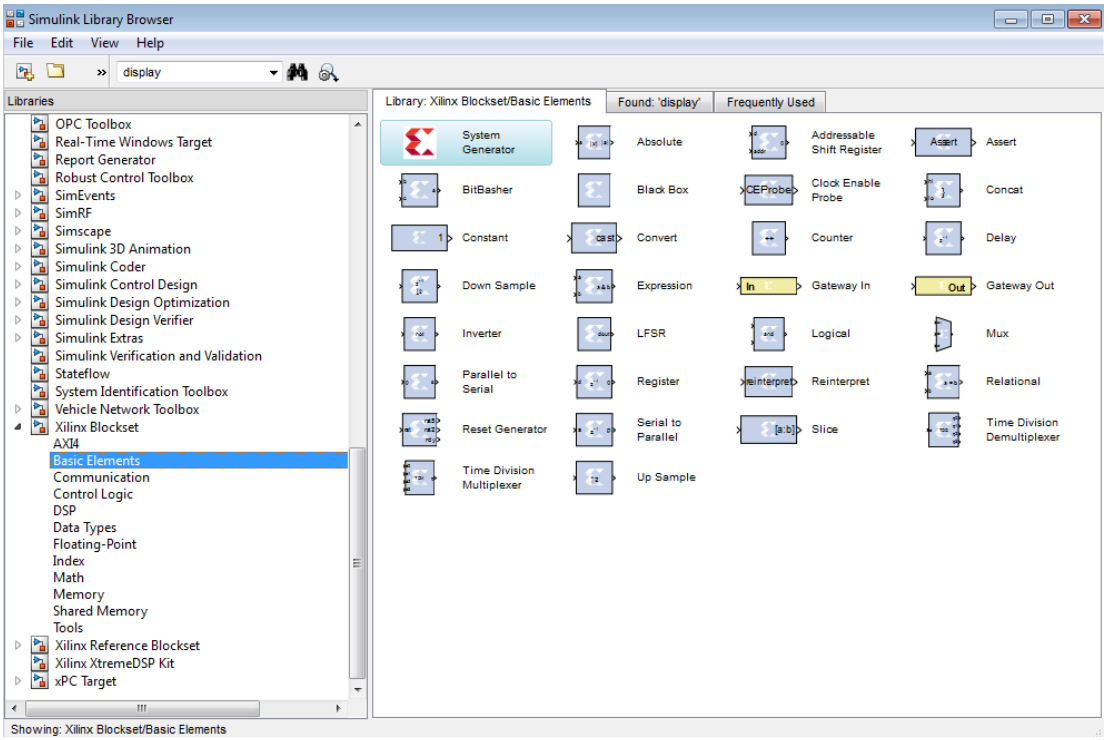
ΠΑΡΑΡΤΗΜΑ Β – Παράδειγμα FPGA In the Loop

Για να μπορεί να υλοποιηθεί και να προσομοιωθεί μια FPGA σχεδίαση σε Simulink πρέπει να γίνει η ένωση με τα εργαλεία της Xilinx (κατασκευαστή του FPGA μας). Αυτό γίνεται με την χρήση του εργαλείου System Generator που μας παρέχει η εταιρεία .Το εργαλείο στην ουσία θα αναλάβει να καλέσει το Matlab,με κατάλληλες παραμέτρους, ώστε να εμπεριέχει όλη την πληροφορία που χρειάζεται για να χειριστεί μια FPGA σχεδίαση. Έτσι όταν θα ανοίξουμε το Matlab Simulink θα δούμε να υπάρχουν επιπλέον κάποια Xilinx blocks και μέσω αυτών και του System Generator block θα γίνει η σύνδεση με το FPGA, όπως θα δούμε και στο παρακάτω παράδειγμα. Για αρχή όμως χρειάζεται να συνδέσουμε τον System Generator με την εγκατάσταση του Matlab που χρησιμοποιούμε. Αυτό γίνεται με την χρήση του System Generator Matlab Configurator όπως φαίνεται παρακάτω :



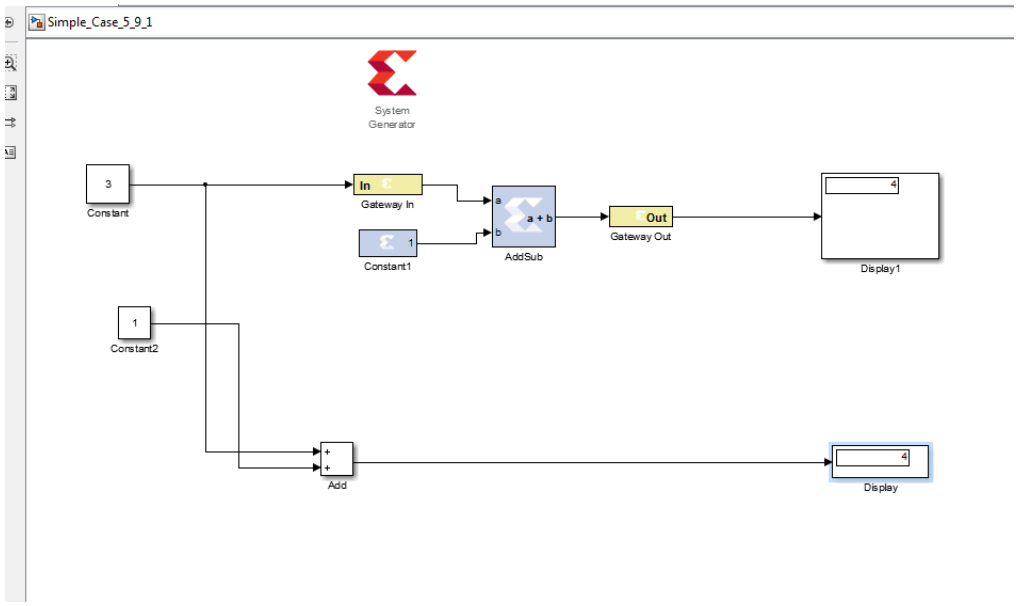
Εικόνα 65 System Generator Matlab Configurator

Έχοντας πραγματοποιήσει την σύνδεση σωστά,όταν μεταφερθούμε σε περιβάλλον Simulink θα δούμε επιπλέον διαθέσιμες βιβλιοθήκες της Xilinx όπως φαίνεται παρακάτω :



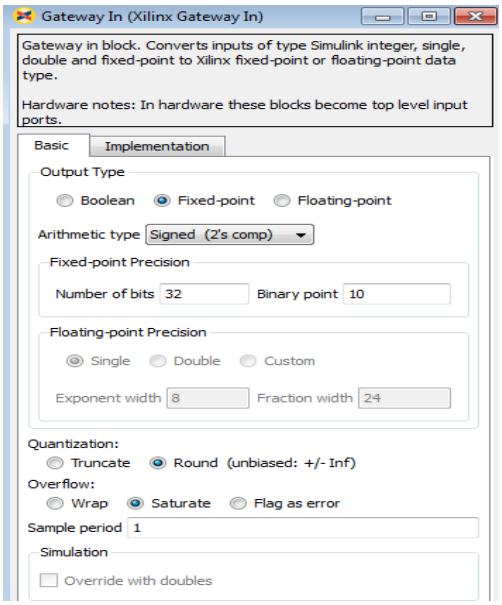
Εικόνα 66 Βιβλιοθήκες Xilinx για την σχεδίαση FPGA

Μια πολύ απλή σχεδίαση για την πρόσθεση 2 αριθμών, τόσο σε FPGA όσο και σε απλή Simulink μορφή, φαίνεται παρακάτω :



Εικόνα 67 Πρόσθεση 2 αριθμών σε Simulink και FPGA

Όπως φαίνεται τα χρωματιστά block αποτελούν block που θα εκτελεστούν στο FPGA. Στο πάνω μέρος της σχεδίασης τοποθετείται υποχρεωτικά το System Generator block καθώς αυτό αναλαμβάνει την σύνδεση με το FPGA. Τα Gateway In και Out μπλοκ είναι η είσοδος και η έξοδος του FPGA και δέχονται σειριακά δεδομένα. Σε αυτά τα block μπορούμε να καθορίσουμε τον τύπο των δεδομένων εισόδου και εξόδου :

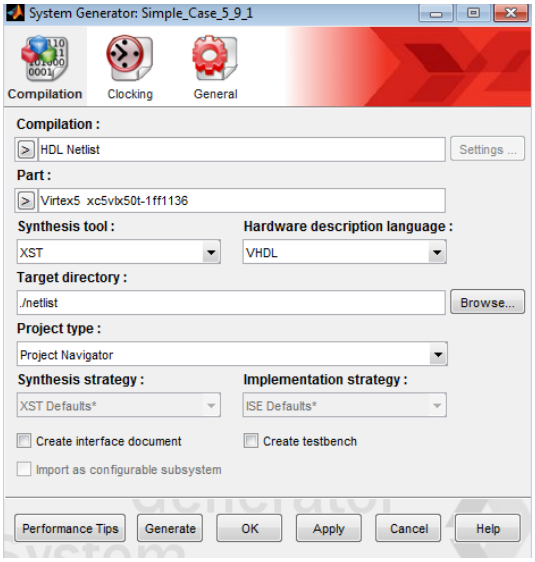


Εικόνα 68 Gateway In Configuration

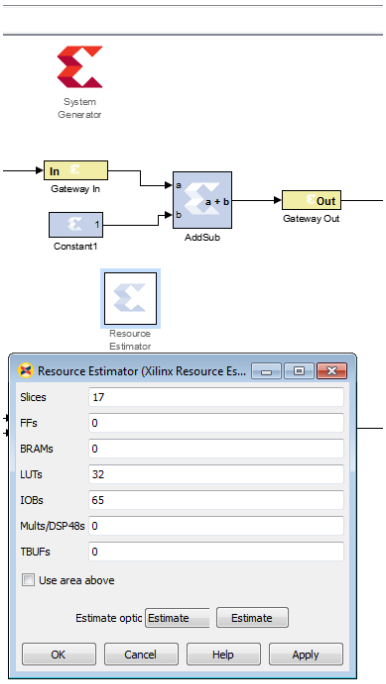
Το Constant block μες στο FPGA λειτουργεί δεσμεύοντας χώρο για μια μεταβλητή και το AddSub block αναλαμβάνει να δημιουργήσει το κύκλωμα της πρόσθεσης. Τέλος η έξοδος επιστρέφεται πίσω στο Simulink από το Gateway Out port και εμφανίζεται σε ένα Display block.

Το τρέξιμο αυτής της σχεδίασης δεν γίνεται πάνω στην πλακέτα απλά προσομοιώνει την λειτουργία της σε αυτή ώστε να επιβεβαιώσουμε ότι λειτουργεί σωστά.

Πριν προβούμε στην εκτέλεση πάνω στο FPGA μπορούμε να πάρουμε μια εκτίμηση για τα resources που καταλαμβάνει η σχεδίαση . Για να γίνει αυτό πρέπει να ανοίξουμε τον System Generator και στο πεδίο Compilation να επιλέξουμε το HDL Netlist και στην συνέχεια να συμπληρώσουμε τα στοιχεία του FPGA που χρησιμοποιούμε και πατάμε OK. Έπειτα προσθέτουμε ένα επιπλέον block στην σχεδίαση, το Resource Estimator block, όπου κάνοντας διπλό κλικ και πατώντας estimate μας δίνονται τα στοιχεία της υλοποίησης πάνω στο FPGA. Αυτή η διαδικασία φαίνεται παρακάτω :



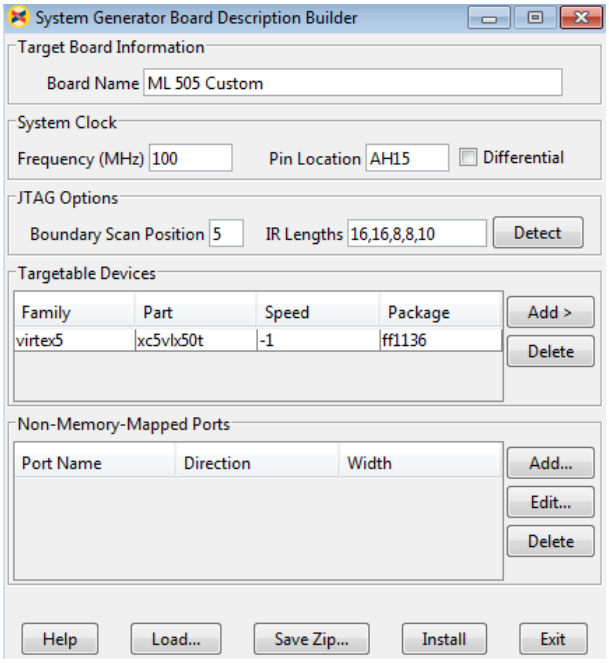
Εικόνα 69 HDL Netlist για το FPGA μας



Εικόνα 70 Resource Estimation της σχεδίασης

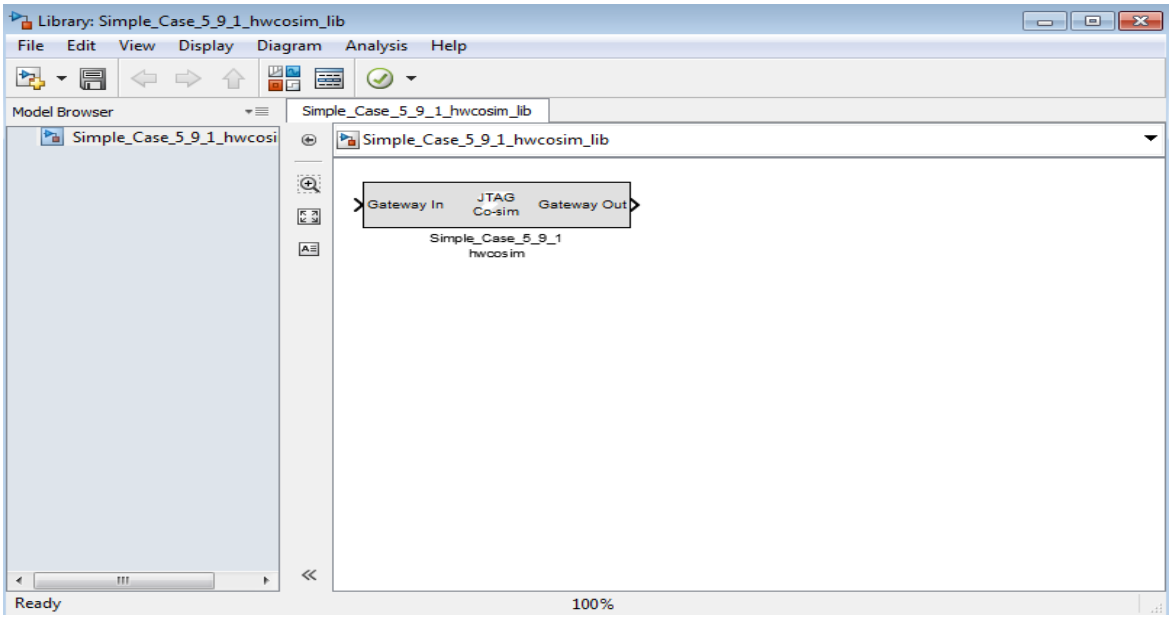
Για να τρέξουμε τώρα αυτή την σχεδίαση σε FPGA πρέπει να ακολουθήσουμε συγκεκριμένα βήματα.

Το πρώτο βήμα είναι να ανοίξουμε το System Generator και στο Compilation πεδίο να επιλέξουμε Hardware Co Simulation όπου εμφανίζονται οι διαθέσιμες πλακέτες για Co Simulation. Αρχικά η πλακέτα μας δεν βρίσκεται στην λίστα οπότε την εισάγαμε manually επιλέγοντας New Compilation Target. Εκεί τοποθετήσαμε τα ζητούμενα στοιχεία:



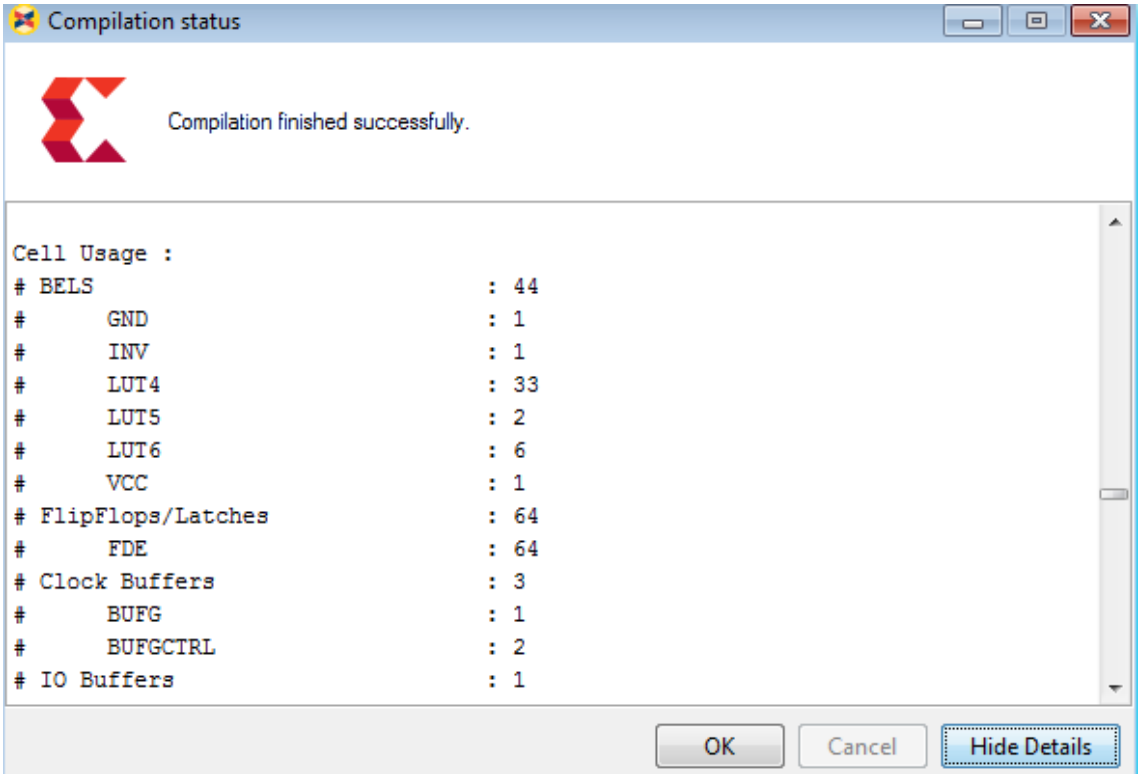
Εικόνα 71 Configuration του FPGA μας για Co Simulation

Έπειτα διαλέγουμε την πλακέτα μας (ML 505 Custom) και πατάμε Generate. Αυτό μας δημιουργεί μια νέα βιβλιοθήκη που εκτελεί την σχεδίαση μας στο FPGA :



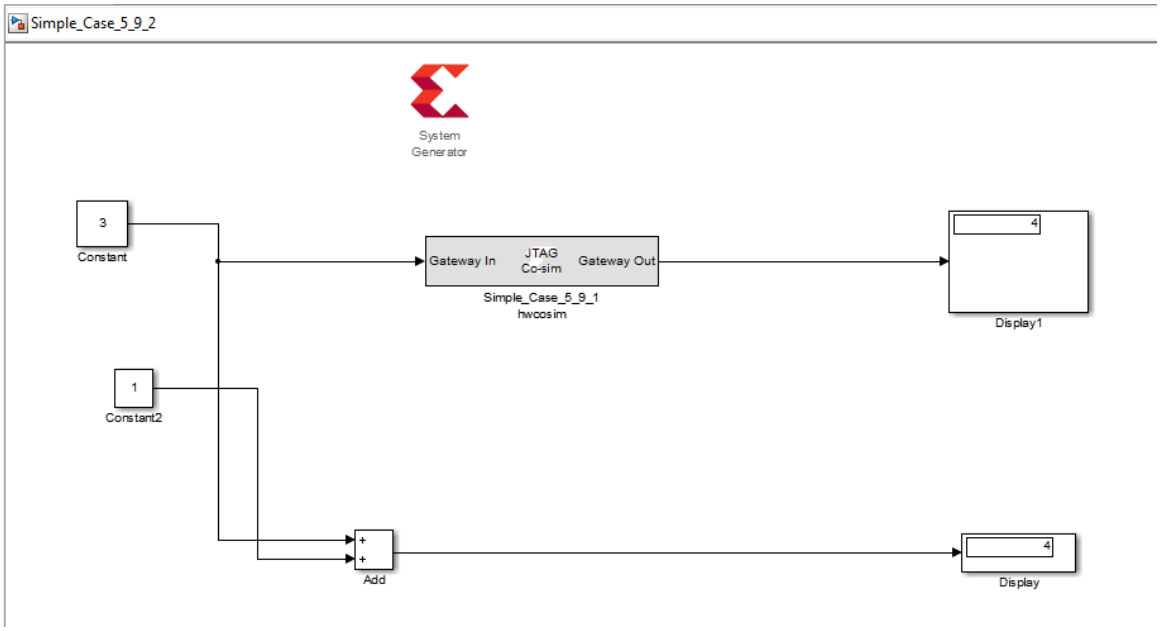
Εικόνα 72 Παραγόμενο block για την σχεδίασή μας

Επιπλέον μπορούμε να επιλέξουμε να δούμε πληροφορίες σχετικά με τα resources που θα καταλαμβάνει η σχεδίαση πάνω στο FPGA :



Εικόνα 73 Επιπλέον λεπτομέρειες για τα resources πάνω στο FPGA

Μέσα σε αυτό το block περιέχεται η λειτουργία της σχεδίασης για το FPGA και μπορούμε πλέον να αντικαταστήσουμε την σχεδίαση με αυτό και να κατεβάσουμε την υλοποίηση στην συσκευή :



Εικόνα 74 Εκτέλεση Simulation με χρήση του FPGA (FPGA In the Loop)

Βιβλιογραφία

[1] Alex Smola, S.V.N. Vishwanathan – Introduction to Machine Learning (Cambridge University Press),2008

[2] Kevin Murphy - Machine Learning: A Probabilistic Perspective(The MIT Press),2012

[3] Ethem Alpaydin – Introduction to Machine Learning SE (The MIT Press),2010

[4] https://en.wikipedia.org/wiki/Machine_learning

[5] Πάνος Αργυράκης, Νευρωνικά Δίκτυα και Εφαρμογές Τόμος Β' ,ΕΑΠ Σχολή Θετικών Επιστημών και Τεχνολογίας,2001

[6] Simon Haykin,Neural Networks – A comprehensive foundation (Pearson Education Inc),1999

[7] https://en.wikipedia.org/wiki/Artificial_neural_network

[8] Kevin Gurney – An introduction to neural networks (UCL Press),1997

[9] Anil K. Jain, Jianchang Mao – Artificial Neural Networks: A Tutorial, Computer - Special issue: neural computing: companion issue to Spring 1996 IEEE Computational Science & Engineering ,Volume 29 Issue 3, March 1996 Page 31-44

[10] Asja Fisher, Christian Igel, "An introduction to Restricted Boltzmann Machines",Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 17th Iberoamerican Congress, CIARP 2012, Buenos Aires, Argentina, September 3-6, 2012. Proceedings . Springer, pp. 14-36. Lecture Notes in Computer Science, vol. 7441

[11] https://en.wikipedia.org/wiki/Boltzmann_machine

[12] http://www.scholarpedia.org/article/Boltzmann_machine

[13] http://deeplearning.net/tutorial/rbm.html

[14] Y. Bengio, "Learning Deep Architectures for AI", Foundations and Trends® in Machine Learning Volume 2 Issue 1, January 2009

[15] N. Le Roux, Y. Bengio, "Representational Power of Restricted Boltzmann Machines and Deep Belief Networks", June 2008, Vol. 20, No. 6, Pages 1631-1649

[16] Geoffrey Hinton, Department of Computer Science Univeristy of Toronto, A Practical Guide to Training Restricted Boltzmann Machines, Version1, 2010

[17] http://blog.echen.me/2011/07/18/introduction-to-restricted-boltzmann-machines/

[18] http://www.scholarpedia.org/article/Deep_belief_networks

[19] Geoffrey Hinton et al,"A fast learning algorithm for deep belief nets", Neural Computation Volume 18 Issue 7, July 2006 ,Pages 1527 – 1554.

[20] http://deeplearning.net/tutorial/DBN.html

[21] Patrick Oliver GLAUNER - Comparison of Training Methods for Deep Neural Networks (MSc), Imperial College London Department of Computing , April 2015

[22] G.E. Hinton and R.R. Salakhutdinov, Reducing the Dimensionality of Data with Neural Networks, Science, 28 July 2006, Vol. 313. no. 5786, pp. 504 - 507.

[23] Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy Layer-Wise Training of Deep Networks, in Advances in Neural Information Processing Systems 19 (NIPS'06), pages 153-160, MIT Press 2007.

[24] Geoffrey Hinton ,2007 Presentation NIPS Tutorial on: Deep Belief Nets

[25] http://www.cs.toronto.edu/~hinton/digits.html

[26] Daniel Ly, Paul Chow, "A High-Performance FPGA Architecture for Restricted Boltzmann Machines", ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '09). pp. 73-82, 2009.

[27] Daniel Ly,Paul Chow,"A High-Performance, Reconfigurable Hardware Architecture for Restricted Boltzmann Machines", IEEE Transactions on Neural Networks. vol 21, no 11. pp. 1780-1792, 2010

[28] Rajat Raina et al,"Large-scale Deep Unsupervised Learning using Graphics Processors" ICML '09 Proceedings of the 26th Annual International Conference on Machine Learning, Pages 873-880

[29] Daniel L. Ly, Volodymyr Paprotski, Danny Yen "Neural Networks on GPUs: Restricted Boltzmann Machines", Department of Electrical and Computer Engineering University of Toronto 2009

[30] Daniel Ly,Paul Chow,"Building a Multi-FPGA Virtualized Restricted Boltzmann Machine Architecture Using Embedded MPI", In ACM International Symposium on Field-Programmable Gate Arrays (FPGA), pages 189-198, February 2011

[31] Sang Kyun Kim et al,"A Highly Scalable Restricted Boltzmann Machine FPGA Implementation", Field Programmable Logic and Applications, 2009. FPL 2009. International Conference Pages 367-372

[32] Sang Kyun Kim et al,"A Large-scale Architecture for Restricted Boltzmann Machines",Proceeding FCCM '10 Proceedings of the 2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines, pages 201-208

[33] http://www.cs.toronto.edu/~hinton/MatlabForSciencePaper.html

[34] http://yann.lecun.com/exdb/mnist/

[35] https://en.wikipedia.org/wiki/Simulink

[36] http://www.mathworks.com/help/simulink/

[37] Eric Peasley, Department of Engineering Science, University of Oxford "An Introduction To Using Simulink", Version 4.0 , 2013

[38] http://www.mathworks.com/help/simulink/ug/types-of-solvers.html

[39] Μ.Ψαράκης,Πανεπιστήμιο Πειραιώς, Σημειώσεις Προηγμένη Ψηφιακή Σχεδίαση

[40] https://en.wikipedia.org/wiki/Field-programmable_gate_array

[41] www.xilinx.com