



Διπλωματική Εργασία
του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και
Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του
Πανεπιστημίου Πατρών

Νικόλαος Αδάλογλου του Μιχαήλ

Αριθμός Μητρώου: 22 7984

Θέμα:

«Αξιοποίηση τεχνικών παράλληλου προγραμματισμού για την
ανάπτυξη
και αξιολόγηση τεχνητών νευρωνικών δικτύων»

Επιβλέπων
Ευάγγελος Δερματάς

Αριθμός Διπλωματικής Εργασίας:

Πάτρα, Σάββατο, 2 Σεπτεμβρίου 2017



ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με θέμα

*«Αξιοποίηση τεχνικών παράλληλου προγραμματισμού για την
ανάπτυξη
και αξιολόγηση τεχνητών νευρωνικών δικτύων»*

Του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας
Υπολογιστών

(Αδάλογλου Νικόλαος του Μιχαήλ)

Αριθμός Μητρώου: 22 7984

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων
Μηχανικών και Τεχνολογίας Υπολογιστών στις
...../...../.....

Ο Επιβλέπων

(Γράφετε ονοματεπώνυμο
και βαθμίδα επιβλέποντος)

Ο Διευθυντής του Τομέα

(Γράφετε ονοματεπώνυμο
και βαθμίδα επιβλέποντος)

Αριθμός Διπλωματικής Εργασίας:

Θέμα: ««Αξιοποίηση τεχνικών παράλληλου προγραμματισμού για την ανάπτυξη και αξιολόγηση τεχνητών νευρωνικών δικτύων»»

Φοιτητής: Αδάλογλου Νικόλαος

Επιβλέπων: Ευάγγελος Δερματάς

Ο κώδικας της παρούσας εργασίας διανέμεται στον παρακάτω σύνδεσμο
github.com/black0017/Adaloglou_Neural_Network

Τα σετ δεδομένων που χρησιμοποιήθηκαν στα πλαίσια της εργασίας ανήκουν στο UCI Machine Learning Repository

[<http://archive.ics.uci.edu/ml>][2013]. Irvine, CA: University of California, School of Information and Computer Science. 



Περίληψη

(Περιγράφεται με λίγα λόγια το αντικείμενο της εργασία σας)

Abstract:



Λεξικό όρων

- ΤΝΔ τεχνητό νευρωνικό δίκτυα , πολυεπίπεδο δίκτυο perceptron
- ΒΡΑ αλγόριθμος οπισθοδρομικής διάδοσης του σφάλματος (Back propagation Algorithm)
- CPU κεντρική μονάδα επεξεργασίας υπολογιστή
- GPU μονάδα επεξεργασίας γραφικών (κάρτα γραφικών)
- FLOPS floating point operation per second

Ευχαριστίες

Ευχαριστώ πολύ όλους εσάς που συμβάλλεται έμπρακτα ώστε να βρίσκομαι στο ατέρμονο μονοπάτι της γνώσης.



Πίνακας περιεχομένων

| | |
|--|----------|
| 1. Νευροεπιστήμη και τεχνητά μοντέλα νευρώνων | 8 |
| Εισαγωγή | 2 |
| Δομή και τύποι νευρώνα..... | 2 |
| Συναρτήσεις Ενεργοποίησης..... | 3 |
| Δίκτυα Εμπρόσθιας Διάδοσης..... | 2 |
| Ο αλγόριθμος οπισθοδρομικής διάδοσης του σφάλματος (BPA) | 3 |
| Αρχιτεκτονικές σχεδιασμού..... | 3 |
| 2. Παράλληλη Επεξεργασία και Cuda | 5 |
| Εισαγωγή στην παράλληλη επεξεργασία..... | 5 |
| Ο πυρήνας και η ιεραρχία νημάτων..... | 5 |
| Ιεραρχία μνήμης..... | 6 |
| Ετερογενής προγραμματισμός | 6 |
| Οδηγίες υψηλών επιδόσεων..... | 6 |
| Μελέτη παραλληλοποίησης πολλαπλασιασμού πινάκων..... | 6 |
| 3. Άλλες Μέθοδοι εκπαίδευσης | 1 |
| Προβλήματα της κλασσικής μεθόδου (global and local strategies) | 24 |
| Οπισθοδρομική διάδοση με αδράνεια..... | 24 |
| Γρήγορη μετάδοση σφάλματος(Qprob) | 3 |
| Ελαστική διάδοση(Rprob) | 3 |
| 4. Υλοποίηση TNΔ σε GPU | 1 |
| Αρχική και γενικευμένη υλοποίηση | 2 |
| Μελέτη παραλληλισμού εμπρόσθιας διάδοσης νευρωνικών δικτύων | 3 |
| Υλοποίηση BPA και παραλλαγών | 3 |
| 5. Εκπαίδευση και Αξιολόγηση TNΔ | 1 |
| Τα dataset που χρησιμοποιήθηκαν | 2 |
| Προβλήματα που αντιμετωπίστηκαν | 2 |
| 6. Αποτελέσματα και συμπεράσματα | 1 |
| Δοκιμές και βελτιστοποιήσεις..... | 2 |
| Σύγκριση αποτελεσμάτων..... | 2 |
| Μελλοντικές βελτιώσεις..... | 2 |
| Παράρτημα Α..... | 1 |



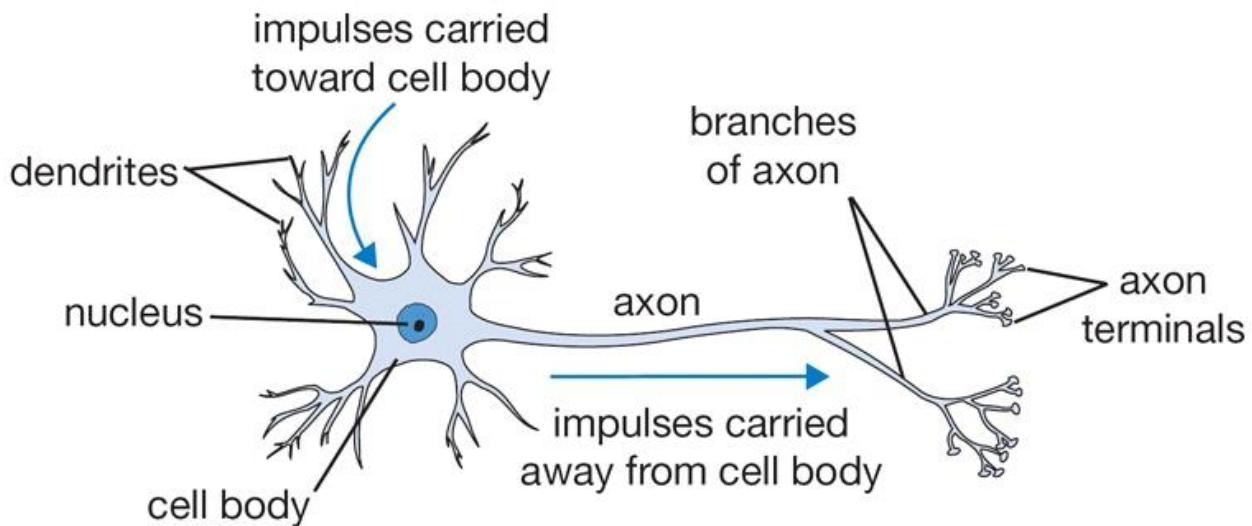
| | |
|---|----------|
| Οδηγίες εγκατάστασης της βιβλιοθήκης Cuda | 2 |
| Λογισμικό Cuda πυρήνων για νευρωνικά δίκτυα | 2 |
| Βιβλιογραφικές Αναφορές..... | 1 |



1. Νευροεπιστήμη και τεχνητά μοντέλα νευρώνων

Εισαγωγή

Ο ανθρώπινος εγκέφαλος είναι φτιαγμένος από ένα απέραντο δίκτυο υπολογιστικών στοιχείων, που καλούνται νευρώνες, συζευγμένα με αισθητήρες υποδοχής και επίδρασης. Ο μέσος ανθρώπινος εγκέφαλος ζυγίζει περίπου 1.3 κιλά και καταλαμβάνει όγκο περίπου 90 κυβικά εκατοστά, περιέχει κοντά στους 100 δισεκατομμύρια κύτταρα διαφόρων τύπων. Ο νευρώνας είναι ένα ειδικό κύτταρο δρά ως μέσω μεταφοράς των εισερχόμενων ηλεκτρικών σημάτων. Ο εγκέφαλος ενός ανθρώπου έχει 10 δισεκατομμύρια νευρώνες. Τα υπόλοιπα 90 δισεκατομμύρια κύτταρα λειτουργούν ως υποστηρικτικά κύτταρα για τους νευρώνες. Το μέγεθος του νευρώνα δεν ξεπερνά το ένα εκατοστό του σημείου στίξης της τελείας. Η αλληλεπίδραση μεταξύ τους επιτυγχάνεται μέσω των επαφών που καλούνται συνάψεις. Κατά μέσο όρο κάθε νευρώνας δέχεται ηλεκτρικά σήματα από χιλιάδες συνάψεις.



Ο εγκέφαλος οργανώνει όλους αυτούς τους νευρώνες με ελάχιστη υπολογιστική ισχύ, σε ένα μαζικά τεράστιο και πολύπλοκο παράλληλο δίκτυο. Στο τελευταίο, οι νευρώνες αλληλεπιδρούν δυναμικά με αποτέλεσμα να παράγουν έναν πανίσχυρο επεξεργαστή πληροφορίας. Οι βιολογικοί νευρώνες είναι 6 με 7 τάξης μεγέθους ποσοτικά πιο αργές από τις τρέχουσες πύλες πυριτίου. Ο εγκέφαλος επεξεργάζεται σήματα μέχρι 40 Hz (κύκλους το δευτερόλεπτο), ενώ ένας σύγχρονος επεξεργαστής είναι 2 με 3 GHz. Είναι φανερό ότι ο σύγχρονος υπολογιστής ξεπερνάει σε επιδόσεις τον άνθρωπο σε προ-προγραμματισμένους και επαναληπτικούς υπολογισμούς. Ωστόσο, σε διαδικασίες πραγματικού χρόνου όπως η κατανόηση ομιλίας και η οπτική αντίληψη, οι οποίες ο άνθρωπος φέρει εις πέρας χωρίς καθόλου προσπάθεια, είναι ακόμη μακριά από το εύρος των σειριακών ψηφιακών υπολογιστών, όσο και αν έχει αυξηθεί η ταχύτητα τους.



Πολλές διαδικασίες είναι εξαιρετικά δύσκολες για σειριακούς υπολογιστές, είτε επειδή ο υπολογιστικός φόρτος απαιτεί ταχύτητες και ποσά μνήμης μη αντιληπτά από την τρέχουσα τεχνολογία , είτε εξαιτίας της πιθανής εγγενής δυσκολίας ορισμένων προβλημάτων, συμπεριλαμβανομένης της ολοκληρωτικής και ακριβής συμβολικής περιγραφής όλων αυτών των προβλημάτων.

Η δυνατότητα των ανθρώπινων όντων να αντιμετωπίζουν τα πρακτικά προβλήματα ασυνείδητα και με σχετική ευκολία σε σχέση με τους σειριακούς υπολογιστές. Το κίνητρο έρευνας των τεχνητών νευρωνικών δικτύων (ΤΝΔ) είναι η πεποίθηση ότι οι ανθρώπινες δυνατότητες σε εργασίες(tasks) πραγματικού χρόνου, όπως η οπτική αντίληψη, η επεξεργασία πληροφορίας των αισθήσεων, η προσαρμοστικότητα όπως και η ευφυής λήψη αποφάσεων γενικότερα , προέρχεται από τις οργανωτικές και υπολογιστικές αρχές που εκθέτει το υπερβολικά περίπλοκο νευρωνικό δίκτυο του εγκεφάλου. Προσδοκίες ταχύτερων και καλύτερων λύσεων μας προκαλούν να φτιάξουμε μηχανές χρησιμοποιώντας παρόμοιες υπολογιστικές και οργανωτικές αρχές, απλοποιημένες και εμπνευσμένες από την βιολογική νευροεπιστήμη του εγκεφάλου.

Για την κατανόηση της λειτουργίας του εγκεφάλου , οι επιστήμονες προσπάθησαν να καταλάβουν τα χαρακτηριστικά απόκρισης ενός μόνο νευρώνα η τις συνδέσεις μιας υπό-ομάδας νευρώνων που σχηματίζουν μεμονωμένες υπό-περιοχές του εγκεφάλου. Ακολουθείται μια στρατηγική από κάτω προς τα πάνω(bottom up). Η μελέτη των νευρώνων σε μοντέλα μικρότερης κλίμακας βοήθησε στο να βρεθεί ποιες ιδιότητες των βιολογικών νευρώνων είναι πιο σημαντικές για μεγαλύτερης κλίμακας δίκτυα όπως του εγκεφάλου. Επιπλέον, τέτοιες προσεγγίσεις καθοδηγούν σε καλύτερους τρόπους εξαγωγής χαρακτηριστικών και ιδιοτήτων για το μοντέλα υψηλού επιπέδου.

Ο τελικός στόχος στην έρευνα των ΤΝΔ είναι η καλύτερη κατανόηση των λειτουργιών και της νοημοσύνης του ανθρώπινου εγκεφάλου. Η επιστήμη αυτήν την στιγμή δεν έχει πλησιάσει ακόμα αυτόν τον απώτερο σκοπό.

Μερικές επικρατούσες οργανωτικές και υπολογιστικές αρχές του εγκεφάλου, όπως έχουν εξαχθεί από την υπάρχουσα ανατομία και φυσιολογία είναι οι παρακάτω:

1. **Μαζικός παραλληλισμός.** Ένας μεγάλος αριθμός από απλές και αργές υπολογιστικές μονάδες, που οργανώνονται συλλογικά για να λύσουν ένα πρόβλημα, με κάθε μονάδα να λειτουργεί ανεξάρτητα. Γι' αυτόν ακριβώς τον λόγο η ανάπτυξη του λογισμικού της εργασίας αυτής πραγματοποιείται με τεχνικές παράλληλου προγραμματισμού σε κάρτα γραφικών.
2. **Υψηλός βαθμός συνδετικής πολυπλοκότητας.** Οι βιολογικοί νευρώνες έχουν τεράστιο αριθμό συνδέσεων με περίπλοκα μοτίβα διασύνδεσης. Τελικά ο εγκέφαλος έχει μεγάλο αριθμό μεταβλητών!
3. **Εκπαιδευσιμότητα (trainability).** Διανευρωνικές παράμετροι αλληλεπίδρασης, όπως το είδος και η ισχύς μιας σύνδεσης, είναι μεταβαλλόμενα σαν αποτέλεσμα της συσσωρευόμενης εμπειρίας από τις αισθήσεις.
4. **Δυαδικές καταστάσεις και συνεχείς μεταβλητές.** Κάθε μονάδα νευρώνα, όπως τα περισσότερα συστήματα, έχει τουλάχιστον δυο καταστάσεις: ηρεμίας και πόλωσης. Ωστόσο, οι μεταβλητές όπως η ιονική και χημική πυκνότητα αλλάζουν συνεχώς χρονικά και χωρικά.
5. **Πολλαπλούς τύπους νευρώνων και βιοσημάτων.**
6. **Περίπλοκη αλληλεπίδραση σημάτων.** Η αλληλεπίδραση από τα σήματα εισόδου ενός νευρώνα είναι εξαιρετικά μη γραμμική και εξαρτάται από πολλούς παράγοντες.



7. **Φυσική αποσύνθεση.** Ο ανθρώπινος εγκέφαλος οργανώνεται σε υποδίκτυο αποτελείται από μερικές χιλιάδες πυκνά συνδεδεμένους νευρώνες. Συνδέσεις σε απόμακρους νευρώνες θεωρούνται ως περισσότερο αραιές, με ελάχιστη ανάδραση. Με αυτόν τον τρόπο, πραγματώνεται η αυτόνομη τοπική και συλλογική επεξεργασία με παράλληλο τρόπο, που ακολουθείται από μια πιο σειριακή και οολοκληρωμένη επεξεργασία των τοπικών και συλλογικών αποτελεσμάτων.
8. **Λειτουργική αποσύνθεση.** Ο εγκέφαλος είναι από λειτουργικής πλευράς, είναι μια αποσύνθεση μιας συλλογής περιοχών. Κάθε περιοχή, είναι υπεύθυνη για συγκεκριμένες εργασίες.

Ο σκοπός της εργασίας είναι η ποιοτική κατανόηση των δυναμικών, της πολυπλοκότητας και των συναρτήσεων δικτύων με τα διάφορα μοντέλα νευρώνων. Η μελέτη εστιάζεται στην ανάλυση, την σύνθεση και την ανάπτυξη ΤΝΔ με σκοπό την επίλυση πρακτικών προβλημάτων. Να σημειωθεί, ως μια απλή αναλογία, ότι τα αεροπλάνα πετούν χωρίς να κινούν τα φτερά τους αλλά εμπνευσμένα από τα πτηνά. Χρησιμοποιούν της ίδιες αεροδυναμικές αρχές για τον έλεγχο της πτήσης. Έτσι, η μαθηματική ανάλυση των απλοποιημένων μοντέλων των ΤΝΔ μας βοηθά να εξάγουμε της βασικές υπολογιστικές αρχές που χρησιμοποιεί αυτή η μαζικά παράλληλη μηχανή.

Ένα μείζον ζήτημα των ΤΝΔ είναι το μέγεθος μαζί με την ιδιότητα της κλιμάκωσης. Μικρά και απλά δίκτυα έχουν υπερβολικά περιορισμένες δυνατότητες. Επομένως, είναι επιθυμητό να ερευνηθεί η αρχιτεκτονική ανάλυσης και σύνθεσης μεγάλων ΤΝΔ με σκοπό να αναδειχτούν οι πιθανές ενδιαφέρον συμπεριφορές και δυνατότητες. Κάτι τέτοιο απαιτεί σε βάθος μαθηματική κατανόηση της δυναμικής συμπεριφοράς, της σύγκλισης, του μεγέθους καθώς και θέματα χρονικής και χωρικής πολυπλοκότητας. Όσο πιο μεγάλο το δίκτυο τόση πιο δύσκολη η μελέτη και η ανάλυση του. Μια επικρατούσα ιδέα είναι ότι για να κατανοηθεί κάτι περίπλοκο, πρέπει να καθοριστεί αν μπορεί να αναπαραχθεί με απλούστερα συστατικά, τα οποία μπορούν να μελετηθούν σε βάθος. Γνώση λοιπόν από μικρότερα δίκτυα μας βοηθά να κατανοήσουμε μεγαλύτερα. Πρέπει να γίνει αντιληπτό, ότι υπάρχουν ποιοτικές αλλαγές στην συμπεριφορά καθώς κάποιος προχωρά από μικρά σε μεγαλύτερα και πολυπλοκότερα ΤΝΔ.

Οι περιοχές έρευνας συγκεντρώνονται, από την οπτική δημιουργίας υπολογιστικές μηχανές, στις εξής :

1. *Η ανάπτυξη ισχυρών και ταχύτερων αλγορίθμων των υπαρχόντων δικτύων.*
2. *Η σχεδίαση νέων δικτύων, διαφορετικών τοπολογιών με σχετικούς αλγόριθμους νέων δυνατοτήτων.*
3. *Η ανάπτυξη θεωρητικού υποβάθρου για δημιουργία μεγάλων δικτύων που θα χρησιμοποιούν μικρότερα.*
4. *Η ανάπτυξη μεγάλων ΤΝΔ, που θα αποτελούνται από αραιά συνδεδεμένα υποδίκτυα, των οποίων η σύνδεση θα είναι πιο πυκνή.*
5. *Η αναζήτηση μεγαλύτερων εφαρμογών, που θα αντικατοπτρίζουν την δομή και τα οργανωτική μορφή των ΤΝΔ.*



Οι συλλογικές υπολογιστικές ικανότητες μιας ομάδας νευρώνων είναι η ουσιαστική διαφοροποίηση στον τομέα του neurocomputing, σε αντίθεση με το κλασσικό μοντέλο υπολογιστή ενός κεντρικού επεξεργαστή, που λειτουργεί με σειριακό μοντέλο κανόνων, όπως περιεγράφηκε από τον Von Neumann. Επειδή ο μέσος αναγνώστης εκτιμάται ότι έχει γνώσεις υπολογιστικών συστημάτων, κρίνεται χρήσιμη η επισήμανση των ομοιοτήτων και των διαφορών των TNΔ σε σχέση με τον Von Neumann υπολογιστή.

Πίνακας 1 Ομοιότητες και Διαφορές TNΔ με Von Neumann υπολογιστή

Νευρωνικό Δίκτυο

1. Εκπαιδεύεται από παραδείγματα προσαρμόζοντας την δομή και την ισχύ σύνδεσης

2. Τα στοιχεία μνήμης και επεξεργασίας αλληλοσυνδέονται

3. Παράλληλα και ασύγχρονα

4. Μπορεί να είναι ανεκτικό σε σφάλματα λόγω της κατανεμημένης αναπαράστασης και απώλειες μεγάλης κλίμακας (large scale redundancy)

5. Αυτό-οργάνωση κατά την διάρκεια εκπαίδευσης

6. Προσαρμόσιμη αποθηκευμένη γνώση, η πληροφορία αποθηκεύεται στις διασυνδέσεις των νευρώνων

7. Η επεξεργασία γίνεται με άναρχο τρόπο

8. Ο κύκλος μηχανής, που επηρεάζει την ταχύτητα επεξεργασίας είναι της τάξης των millisecond

Von Neumann υπολογιστής

Προγραμματίζεται ρητά με εντολές βάση λογικής

Διαχωρισμός μνήμης και επεξεργασίας

Ακολουθιακά ή σειριακά, ψηφιακά, συγχρονισμός με ρολόι

Μη ανεκτικά σε λάθη

Εξαρτώμενο από το λογισμικό

Η γνώση είναι αποθηκευμένη στην κύρια μνήμη και αντικαθίσταται

Η επεξεργασία γίνεται με επιτακτικό και αυταρχικό τρόπο

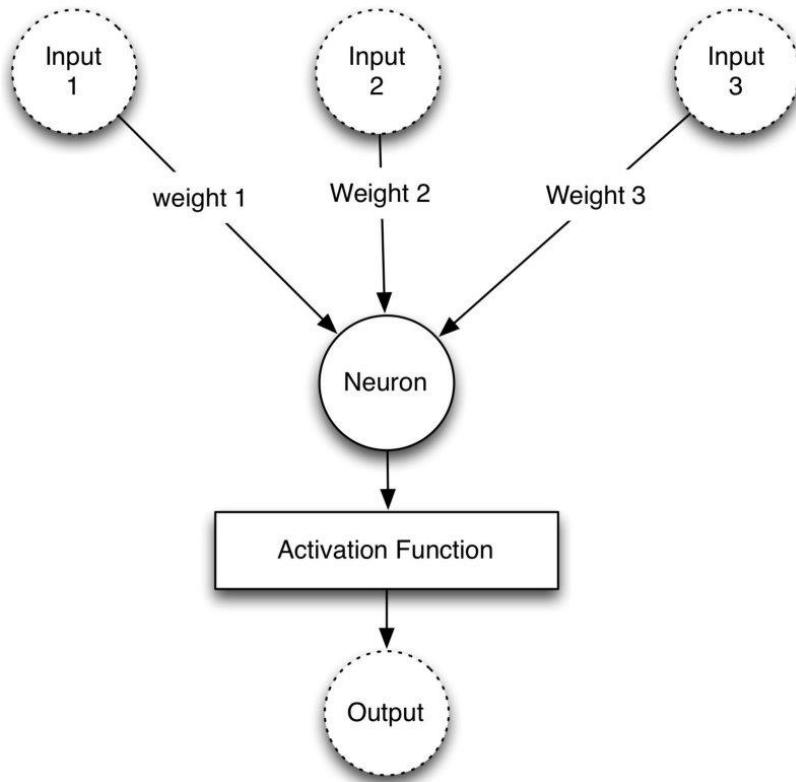
Ο κύκλος μηχανής, που επηρεάζει εντολή ενός προγράμματος στην κεντρική μονάδα επεξεργασίας, είναι της τάξης των nanosecond



Δομή και τύποι νευρώνα

Νευρώνες και Επίπεδα

Όλα τα TNΔ χρησιμοποιούν κάποιο είδος νευρώνα. Υπάρχουν πολλά διαφορετικά είδη νευρωνικών δικτύων και οι προγραμματιστές ανακαλύπτουν συνεχώς νέες πειραματικές δομές νευρωνικών δικτύων. Συνεπώς, δεν είναι δυνατή η κάλυψη κάθε αρχιτεκτονικής. Ωστόσο, υπάρχουν ορισμένες ομοιότητες μεταξύ όλων των υλοποιήσεων. Ένας αλγόριθμος που ονομάζεται TNΔ θα αποτελείται συνήθως από μεμονωμένες, διασυνδεδεμένες μονάδες που να ονομάζονται νευρώνες. Το όνομα για μια μονάδα επεξεργασίας ποικίλλει μεταξύ των πηγών της βιβλιογραφίας (κόμβος, νευρώνας, μονάδα).



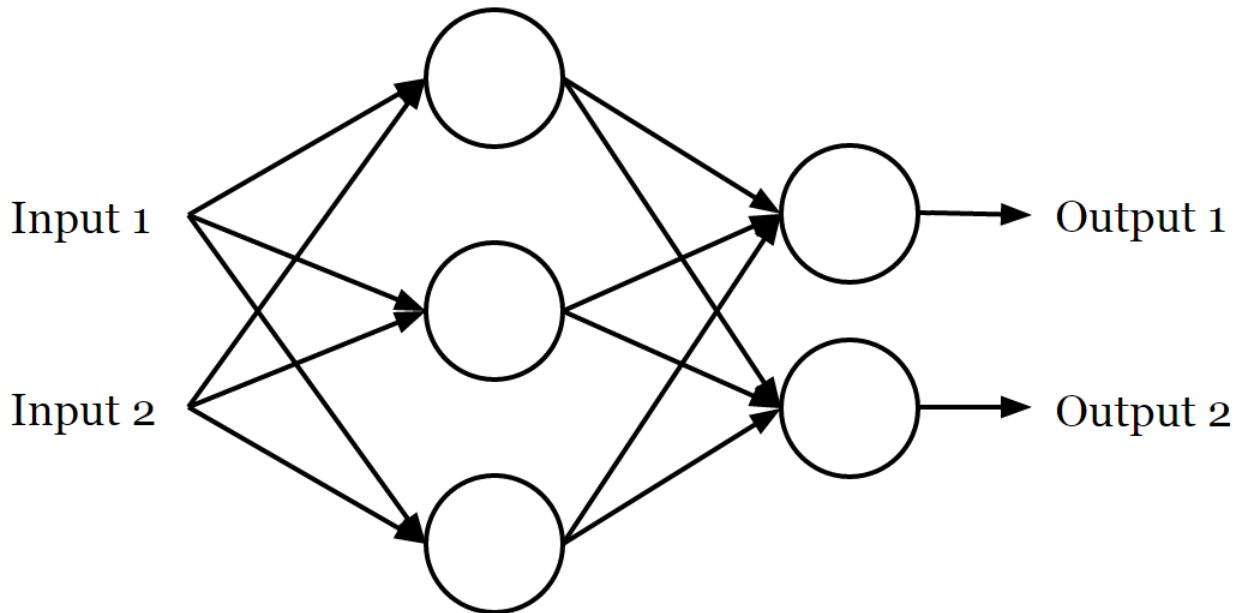
Ο τεχνητός νευρώνας λαμβάνει είσοδο από μία ή περισσότερες 'πηγές' που μπορεί να είναι άλλοι νευρώνες ή δεδομένα που τροφοδοτούνται από ένα πρόγραμμα. Συχνά η είσοδος κωδικοποιείται σε πραγματική μεταβλητή (τύπου float). Κάθε νευρώνας πολλαπλασιάζει κάθε μία από αυτές τις εισόδους με ένα βάρος. Στη συνέχεια προσθέτει τα γινόμενα και μεταδίδει αυτό το άθροισμα σε μια συνάρτηση ενεργοποίησης.

$$f(x_i, w_i) = \phi\left(\sum_i (w_i \cdot x_i)\right)$$



Στην παραπάνω εξίσωση, οι μεταβλητές x και w αναπαριστούν την είσοδο και τα βάρη αντίστοιχα. Ο αριθμός των βαρών οφείλει να είναι πάντα ίδιος με των αριθμό των εισόδων. Κάθε βάρος πολλαπλασιάζεται με την αντίστοιχη είσοδο του και το αποτέλεσμα τροφοδοτείται στην συνάρτηση ενεργοποίησης που δηλώνεται από το ελληνικό γράμμα ϕ . Συνεπώς από τον νευρώνα λαμβάνεται μία μόνο έξοδος.

Χρησιμοποιώντας την παραπάνω σαν στοιχειώδη δομή δημιουργούμε δίκτυα που αποτελούνται από πολλούς νευρώνες, συνδέοντας τους με διαφορετικούς τρόπους. Μια απλή τοπολογία φαίνεται στο σχήμα που ακολουθεί.



Στο σχήμα φαίνονται οι νευρώνες να είναι νοητά διατεταγμένοι σε στρώματα/επίπεδα. Οι νευρώνες εισόδου είναι το πρώτο στρώμα, οι ενδιάμεσοι νευρώνες δημιουργούν το δεύτερο στρώμα, ενώ το τρίτο στρώμα περιέχει άλλους 2 νευρώνες που μας δίνουν τις εξόδους. Ενώ τα περισσότερα TNΔ οργανώνουν τους νευρώνες σε επίπεδα, αυτό δεν είναι απαραίτητη συνθήκη. Αυτή η τοπολογία έρχεται σε αντίθεση με το μοντέλο του ανθρώπινου εγκεφάλου, αλλά προγραμματιστικά μας διευκολύνει για μια πληθώρα λόγων που θα αναλυθούν παρακάτω. Επίσης, να σημειωθεί πως ο εγκέφαλος δεν διατηρεί πλήρεις συνδέσεις όπως τα TNΔ που θα αναφερθούμε. Το 2002 o Stanley εισήγαγε μια αρχιτεκτονική που ονομάζεται Neuroevolution of Augmenting Topologies (NEAT). Τα NEAT νευρωνικά μπορούν να έχουν μια πολύ μπερδεμένη, μη στρωματοποιημένη αρχιτεκτονική. Τέλος, αξίζει να αναφερθεί ότι μεγάλη έρευνα γίνεται και στην δημιουργία νέων τοπολογιών από διαγράμματα Voronoi [link]. Στα πλαίσια της εργασίας, θα αναφερθούν τεχνικές σχεδιασμού μόνο για δίκτυα οργανωμένα σε επίπεδα.

Οι νευρώνες που σχηματίζουν ένα επίπεδο μοιράζονται αρκετά χαρακτηριστικά. Πρώτον, κάθε νευρώνας σε ένα επίπεδο έχει την ίδια συνάρτηση ενεργοποίησης. Ωστόσο, τα επίπεδα μπορεί να έχουν διαφορετικές συναρτήσεις ενεργοποίησης. Δεύτερον, τα επίπεδα είναι πλήρως συνδεδεμένα με τα επόμενα. Με άλλα λόγια, κάθε νευρώνας σε ένα στρώμα συνδέεται με όλους τους νευρώνες στο προηγούμενο στρώμα. Στην τοπολογία αυτήν αναφερόμαστε ως πλήρως διασυνδεδεμένα TNΔ.



Τα περισσότερα δίκτυα θα έχουν μεταξύ ενός και τριών κρυμμένων επιπέδων. Εκτός από τις στρατηγικές βαθιάς μάθησης(deep learning), δίκτυα με περισσότερα από δύο κρυφά στρώματα είναι σπάνια.

Ενδέχεται επίσης να παρατηρήσετε ότι τα βέλη του σχήματος πάντοτε δείχνουν από την είσοδο στην έξοδο, δηλαδή προς μια κατεύθυνση. Αυτός ο τύπος δικτύου ονομάζεται δίκτυο εμπρόσθιας διάδοσης(feedforward neural network). Υπάρχουν και τοπολογίες με επαναλαμβανόμενα TNΔ που σχηματίζουν ανεστραμμένους βρόχους μεταξύ των νευρώνων.

Τύποι νευρώνων

Χωρίζουμε τους νευρώνες σε 4 κατηγορίες ανάλογα με την θέση τους στην τοπολογία.

- Εισόδου και εξόδου. Σχεδόν κάθε δίκτυο έχει νευρώνες εισόδου και εξόδου. Οι νευρώνες εισόδου δέχονται δεδομένα από το πρόγραμμα. Οι νευρώνες εξόδου παρέχουν επεξεργασμένα δεδομένα πίσω στο πρόγραμμα. Ομαδοποιούνται σε ξεχωριστά επίπεδα που ονομάζονται στρώματα εισόδου και εξόδου. Συνήθως αναπαρίστανται με ένα διάνυσμα πραγματικών αριθμών.
- Κρυφοί νευρώνες. Έχουν δύο σημαντικά χαρακτηριστικά. Πρώτον, λαμβάνουν μόνο εισόδους από άλλους νευρώνες, όπως το διάνυσμα εισόδου ή άλλους κρυμμένους νευρώνες. Δεύτερον, παράγουν έξοδο μόνο σε άλλους νευρώνες, όπως η νευρώνες εξόδου ή άλλοι κρυμμένοι νευρώνες. Βοηθούν το TNΔ να μοντελοποιήσει την είσοδο. Η κοινή ερώτηση των προγραμματιστών αφορά τον αριθμό των κρυμμένων νευρώνων σε ένα δίκτυο. Η απάντηση στο ερώτημα αυτό είναι αρκετά πολύπλοκη, παρόλα αυτά ερευνητές έχουν αποδείξει ότι ένα TNΔ με ένα απλό κρυφό στρώμα μπορεί να προσεγγίσει οποιαδήποτε συνάρτηση, αρκεί να έχει αρκετούς νευρώνες.
- Νευρώνες πόλωσης(bias). Οι προγραμματιστές προσθέτουν νευρώνες πόλωσης. Λειτουργούν σαν νευρώνες εισόδου που παράγουν πάντα την τιμή 1, γι' αυτό δεν συνδέονται με το προηγούμενο επίπεδο. Δεν έχουν όλα τα TNΔ bias νευρώνες. Τα βάρη του νευρώνα επιτρέπουν την προσαρμογή της κλίσης ή την μορφή της συνάρτησης ενεργοποίησης. Ο bias νευρώνας μετατοπίζει την σιγμοειδή καμπύλη, η οποία όταν το διάνυσμα εισόδου είναι κοντά στο 0, χωρίς την προσθήκη του bias, οδηγείται στο 0.5.

$$f(x, w, b) = \frac{1}{1 + e^{-(wx+b)}}$$



Συναρτήσεις ενεργοποίησης νευρώνα

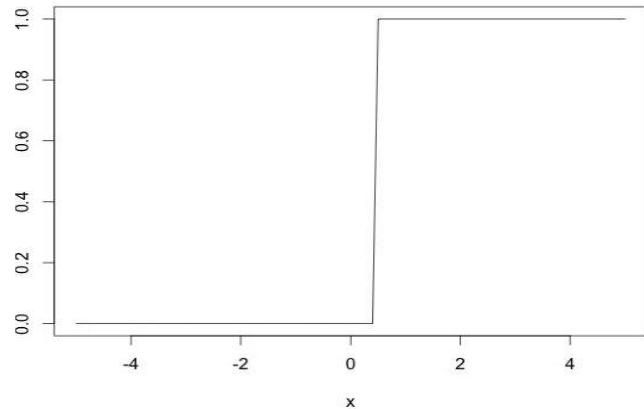
Στον προγραμματισμό ΤΝΔ, οι συναρτήσεις ενεργοποίησης ή μεταφοράς καθορίζουν όρια για την έξοδο των νευρώνων. Τα νευρικά δίκτυα μπορούν να χρησιμοποιούν πολλές διαφορετικές συναρτήσεις ενεργοποίησης. Θα συζητηθούν εδώ οι πιο κοινές. Η επιλογή μιας συνάρτησης ενεργοποίησης για το ΤΝΔ είναι μια σημαντική παρατήρηση, επειδή μπορεί να επηρεάσει τον τρόπο με τον οποίο πρέπει να μορφοποιήσετε τα δεδομένα εισόδου κατά την προ-επεξεργασία.

Η συνάρτηση ενεργοποίησης-συμβολίζεται με το γράμμα ϕ -είναι ένας μη γραμμικός τελεστής και εφαρμόζεται στην έξοδο κάθε νευρώνα. Ακολουθεί τα χαρακτηριστικά των σιγμοειδών συναρτήσεων (sigmoid), δηλαδή να είναι γνησίως αύξουσα, να ορίζεται στο σύνολο των πραγματικών αριθμών και να έχει φραγμένο πεδίο τιμών.

Βηματική συνάρτηση ενεργοποίησης

Η πρώτη μη γραμμική συνάρτηση ενεργοποίησης που χρησιμοποιήθηκε το 1943 από τους McCulloch & Pitts ονομάζεται βηματική ή συνάρτηση κατωφλίου. Οι νευρώνες παλιότερα ονομάζονταν perceptron και αναπτύχθηκαν με αυτήν την απλή συνάρτηση, στην οποία η έξοδος είναι δυαδική, συγκρινόμενη κάθε φορά με το 0.5.

$$\phi(x) = \begin{cases} 1, & \text{if } x \geq 0.5. \\ 0, & \text{otherwise.} \end{cases}$$

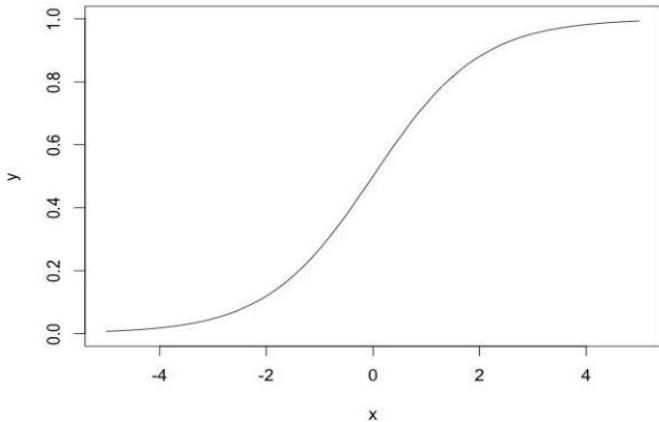


Εκθετική σιγμοειδής συνάρτηση (logistic)

Η Εκθετική σιγμοειδής ή logistic είναι μια συνήθης επιλογή για εμπρόσθια πλήρως διασυνδεδεμένα νευρωνικά δίκτυα που η έξοδος είναι μόνο θετικοί αριθμοί. Παρά τη διαδεδομένη χρήση της, η υπερβολική εφαπτομένη είναι συνήθως μια καταλληλότερη επιλογή.

$$\phi(x) = \frac{1}{1 + e^{-x}} \quad \phi'(x) = \phi(x)(1 - \phi(x))$$





Υπερβολική εφαπτομένη (tanh)

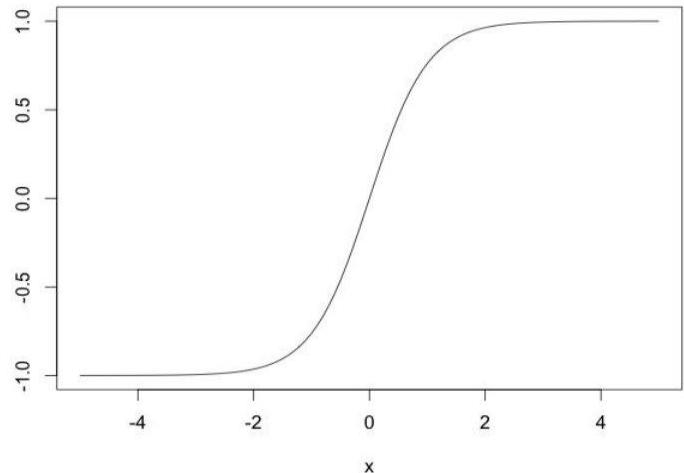
Μια όλο και πιο διαδεδομένη συνάρτηση για ΤΝΔ. Το μεγαλύτερο πεδίο τιμών και το γεγονός ότι παίρνει και αρνητικές τιμές της δίνει σοβαρό πλεονέκτημα σε σχέση με την εκθετική σιγμοειδή. Περισσότερες συγκρίσεις θα αναφερθούν στο κεφάλαιο της εκπαίδευσης με BPA(back propagation algorithm). Τονίζεται ότι ο όρος υπερβολική εφαπτομένη είναι καταχρηστικός και συχνά συγχέεται με την γνωστή μαθηματική συνάρτηση αλλά αυτή που χρησιμοποιείται σε ΤΝΔ είναι διαφορετική.

$$\tanh(x) = \frac{e^{ax} - e^{-ax}}{(1-e^{-2x})} - 1 \neq \frac{e^{ax} - e^{-ax}}{e^{ax} + e^{-ax}} = \text{HyperTan}(x).$$

Ακολουθεί και σχετική σχηματική αναπαράσταση :

$$\phi(x) = \tanh(x)$$

$$\phi'(x) = 1.0 - \phi^2(x)$$



Συνάρτηση διορθωμένης γραμμικής μονάδας (ReLU)

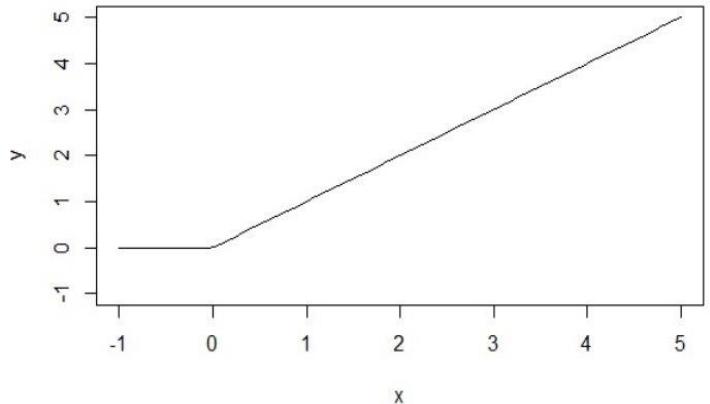
Θα εξεταστεί γιατί η ReLU αποδίδει συχνά καλύτερα από άλλες συναρτήσεις ενεργοποίησης για κρυφά επίπεδα. Ο βασικός λόγος της αυξημένης απόδοσης οφείλεται στο γεγονός ότι η ReLU είναι μια γραμμική συνάρτηση μη κορεσμού. Ο κορεσμός είναι το μεγαλύτερο πρόβλημα των δυο προηγούμενων σιγμοειδών συναρτήσεων. Σε αντίθεση λοιπόν με την logistic ή tanh, η ReLU δεν κορέζεται στο -1, 0 ή 1. Οι πιο πρόσφατες έρευνες αναφέρουν ότι τα κρυμμένα επίπεδα του ΤΝΔ



πρέπει να χρησιμοποιούν την ενεργοποίηση του ReLU. Το σχήμα που ακολουθεί δείχνει το γράφημα της συνάρτησης ενεργοποίησης ReLU:

$$\phi(x) = \max(0, x)$$

$$\frac{dy}{dx} \phi(x) = \begin{cases} 1 & x > 0 \\ 0 & x \leq 0 \end{cases}$$



Συνάρτηση softmax

Η τελευταία συνάρτηση που θα εξεταστεί είναι η softmax. Μαζί με τη συνάρτηση γραμμικής ενεργοποίησης $\varphi(x) = x$, η softmax βρίσκεται συνήθως στο στρώμα εξόδου ενός TNΔ.

Χρησιμοποιείται σε ένα δίκτυα ταξινόμησης. Ο νευρώνας που έχει την υψηλότερη τιμή κάνει την είσοδο ως μέλος της κλάσης του. Είναι μια προτιμώμενη μέθοδος, διότι η softmax αναγκάζει την έξοδο του νευρωνικού να αναπαριστά την πιθανότητα να πέσει η είσοδος σε κάθε μία από τις κλάσεις. Χωρίς τη softmax, οι έξοδοι του νευρώνα είναι απλά αριθμητικές τιμές, με την υψηλότερη ένδειξη της επικρατούσας κλάσης.

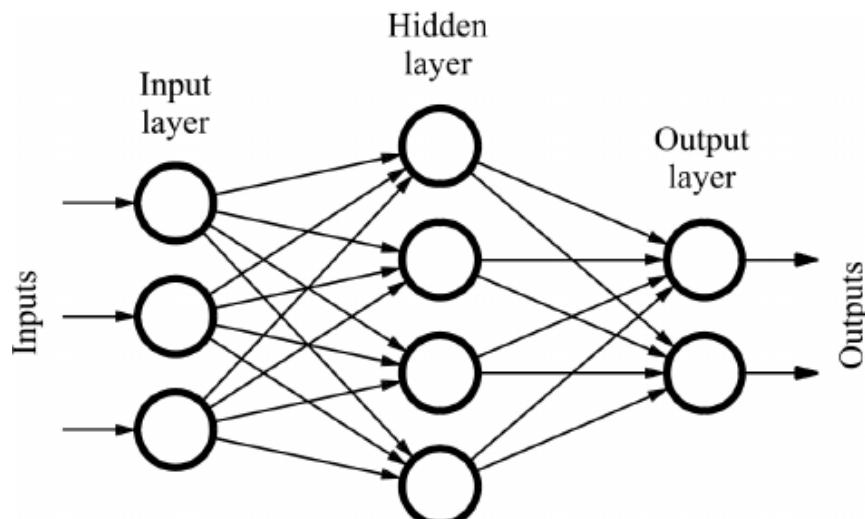
$$\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}} \quad \frac{\partial \phi_i}{\partial z_i} = \phi_i(1 - \phi_i)$$



Δίκτυα Εμπρόσθιας Διάδοσης (feedforward networks)

Τα νευρωνικά δίκτυα εμπρόσθιας διάδοσης είναι από τους πιο συνηθισμένους αλγορίθμους στην τεχνητή νοημοσύνη. Θα αναφερθούμε μόνο στα πολυεπίπεδα δίκτυα perceptron εμπρόσθιας διάδοσης. Η ευρεία διάδοση των πολυεπίπεδων δικτύων perceptron οφείλεται στην ισχυρή τους πολυπλοκότητα και μη γραμμικότητα να παρέχει δυνατότητες προσομοίωσης με ικανοποιητική ακρίβεια συνεχείς μη γραμμικές διανυσματικές συναρτήσεις, ενώ οι συντελεστές βαρύτητας υπολογίζονται από παραδείγματα. Υπάρχουν και άλλοι αλγόριθμοι όπως τα δίκτυα radial-basis function (RBF), που δεν αποτελούν αντικείμενο μελέτης αυτής της εργασίας.

Λόγω της ευελιξίας που περιγράφηκε, η αρχιτεκτονική αυτή είναι η πιο δημοφιλής. Επομένως, θα διερευνηθεί πώς επιτυγχάνεται εκπαίδευση και πώς επεξεργάζεται ένα πρότυπο/παράδειγμα από το σύνολο των δεδομένων. Ο όρος feedforward περιγράφει τον τρόπο με τον οποίο το δίκτυο επεξεργάζεται τα πρότυπα. Σε ένα feedforward δίκτυο, κάθε επίπεδο συνδέεται πλήρως με το επόμενο. Αυτές οι συνδέσεις εκτείνονται από την είσοδο προς την έξοδο, αλλά δεν υπάρχουν συνδέσεις προς τα πίσω. Το συμπέρασμα επιτρέπει την αναπαράσταση ενός ΤΝΔ με έναν κατευθυνόμενο μη κυκλικό γράφο.



Μπορούμε να εκπαιδεύσουμε τα feedforward δίκτυα με μια ποικιλία τεχνικών από την ευρεία κατηγορία αλγορίθμων backpropagation, μια μορφή επιβλεπόμενης μάθησης που θα συζητηθεί σε επόμενη ενότητα. Σε αυτήν την ενότητα θα επικεντρωθούμε στην αρχικοποίηση των βαρών, τις απαραίτητες κανονικοποιήσεις, καθώς και τις συναρτήσεις υπολογισμού σφάλματος επιβλεπόμενης εκπαίδευσης.

Οι αλγόριθμοι που αναπροσαρμόζουν τα βάρη χρειάζονται μια μετρική σφάλματος για την ποσοτικοποίηση της απόκλισης της εξόδου του ΤΝΔ από την γνωστή επιθυμητή έξοδο. Αυτοί οι αλγόριθμοι βελτιστοποίησης υπολογισμού έχουν ως στόχο να ελαχιστοποιήσουν την αντικειμενική συνάρτηση του σφάλματος. Αυτό το αποτέλεσμα επιτρέπει σε οποιονδήποτε αλγόριθμο εκπαίδευσης να εκπαιδεύσει το δίκτυο.



Υπολογισμός εξόδου

Αφού έχει εξεταστεί πως υπολογίζεται η έξοδος ενός νευρώνα, μπορούμε να γενικεύσουμε την ίδια διαδικασία για το πολυεπίπεδο δίκτυο perceptron. Αυτή η διαδικασία επιτρέπει τον υπολογισμό των εξόδων από τους νευρώνες εισόδου μέχρι έξοδο.

$$f(x_i, w_i) = \phi\left(\sum_i (w_i \cdot x_i)\right)$$

for each neuron
for each level

Αρχικοποίηση βαρών

Τα βάρη ενός TNΔ καθορίζουν την έξοδο. Η διαδικασία εκπαίδευσης μπορεί να προσαρμόσει αυτά τα βάρη έτσι ώστε να παράγεται μοντελοποίηση μεγάλης απόδοσης. Οι περισσότεροι αλγόριθμοι εκπαίδευσης αρχίζουν με την αρχικοποίηση των βαρών σε τυχαίες τιμές. Η εκπαίδευση προχωρά στη συνέχεια μέσω μιας σειράς επαναλήψεων που προσαρμόζουν τα βάρη για να παράγουν καλύτερη απόδοση. Τα τυχαία βάρη ενός νευρικού δικτύου επηρεάζουν πόσο καλά μπορεί να εκπαιδευτεί. Εάν ένα δίκτυο αποτύχει να μοντελοποιήσει την είσοδο, υπάρχει περίπτωση να οφείλετε στην αρχική κατάσταση των βαρών. Επανεκκινώντας με ένα νέο σύνολο τυχαίων βαρών μπορεί να διορθωθεί το πρόβλημα αυτό. Ωστόσο, αυτή η διαδικασία πειραμάτων και επανεκκινήσεων μπορεί να είναι ψυχοφθόρα και κουραστική όταν δοκιμάζετε διαφορετικές αρχιτεκτονικές, δηλαδή πιθανούς συνδυασμούς κρυφών στρωμάτων και νευρώνων. Κάθε φορά που προσθέστε ένα νέο επίπεδο και βελτιώνεται η απόδοση, δεν είναι σίγουρο αν αυτή η βελτίωση προέκυψε από την αλλαγή που έγινε ή από το νέο σύνολο βαρών.

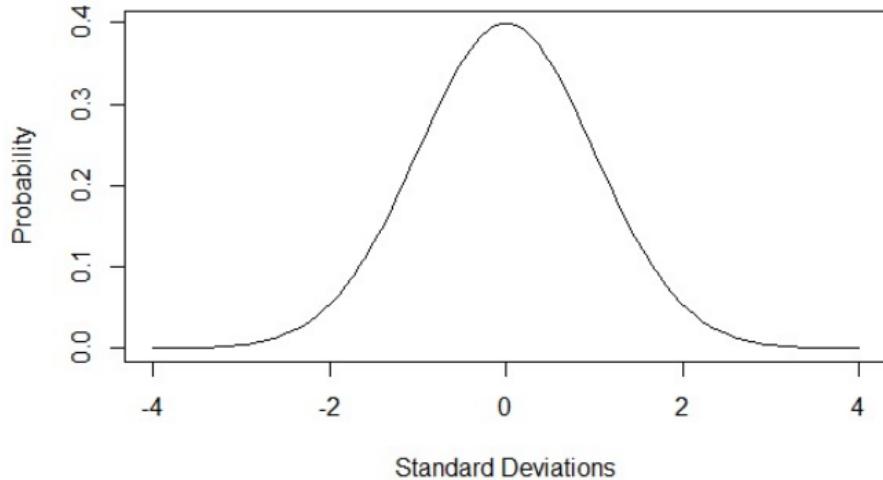
Μία από τις πιο συνηθισμένες αλλά λιγότερο αποτελεσματικές προσεγγίσεις για την αρχικοποίηση του βάρους είναι η επιλογή των τυχαίων βαρών μέσα σε ένα συγκεκριμένο εύρος. Επιλέγονται συνήθως αριθμοί μεταξύ -1 και +1 ή -5 και +5. Εάν θέλετε να διασφαλίσετε ότι θα έχετε το ίδιο σύνολο τυχαίων βαρών κάθε φορά, θα πρέπει να χρησιμοποιήσετε τον ίδιο σπόρο(seed) για την γεννήτρια αριθμών. Ο σπόρος(seed) καθορίζει ένα σύνολο προκαθορισμένων τυχαίων βαρών. Οι τιμές παραμένουν τυχαίες. Δεν μπορείτε να τα προβλέψετε, αλλά θα έχετε πάντα αυτές τις τιμές όταν επιλέγετε το ίδιο seed.

Αυτός είναι ένας καλός τρόπος για να ελέγχετε αν μια αλλαγή στην υλοποίηση του αλγόριθμο ενός TNΔ λειτουργεί όπως είναι επιθυμητό.

Μια άλλη λύση είναι το βάρη να ακολουθούν κανονική κατανομή με μέση τιμή 0 και διασπορά σ^2 . Από την στατιστική, είναι γνωστό ότι το 99.7% των τιμών της κατανομής αυτής θα βρίσκονται στο διάστημα

$$\text{Normal Distribution Range (for } \mu = 0) = \{-3\sigma, +3\sigma\}$$





Υπήρξε αντικείμενο πολλών ερευνητών η επιλογή της διασποράς για την αρχικοποίηση των βαρών. Η πιο επικρατέστερη τεχνική είναι η επιλογή διασποράς Xavier [[link](#)]. Η αρχικοποίηση βάρους Xavier θέτει όλα τα βάρη σε κανονικά κατανεμημένους τυχαίους αριθμούς. Αυτά τα βάρη είναι πάντα κεντραρισμένα στο 0. Ωστόσο, η διασπορά τους ποικίλει ανάλογα με τον αριθμό των συνδέσεων που υπάρχουν για το τρέχον επίπεδο βαρών. Συγκεκριμένα, η ακόλουθη εξίσωση μπορεί να καθορίσει την τυπική απόκλιση, για n_{in} εισόδους και n_{out} εξόδους:

$$Var(W) = \frac{2}{n_{in} + n_{out}}$$

Η παραπάνω εξίσωση δείχνει τον τρόπο υπολογισμού της διακύμανσης για όλα τα βάρη. Η τετραγωνική ρίζα της διακύμανσης είναι η τυπική απόκλιση. Συνήθως χρειάζεται να πάρετε την ρίζα της παραπάνω εξίσωσης. Διευκρινίζεται ότι αυτή η διαδικασία αρχικοποίησης γίνεται για κάθε επίπεδο του εμπρόσθιου ΤΝΔ.



Υπολογισμός σφάλματος

Παρακάτω αναφέρονται οι πιο γνωστοί τρόποι υπολογισμού σφάλματος που χρησιμοποιούνται για την ποσοτικοποίηση του σφάλματος μεταξύ του επιθυμητού διανύσματος (y) και του διανύσματος εξόδου που προκύπτει από το μοντέλο (\hat{y})

1. Μέσο τετραγωνικό σφάλμα: χρησιμοποιείται σε πολλά επιστημονικά πεδία είναι πάντοτε ποσότητα μη αρνητική. Μετράει το μέσο όρο της διαφοράς των δυο διανυσμάτων υψωμένη στο τετράγωνο. Με την ύψωση στο τετράγωνο εξασφαλίζεται ότι η συνάρτηση σφάλματος θα έχει μοναδικό ελάχιστο. Σημειώνεται, ότι το τελικό άθροισμα διαιρείται με τον μέγεθος του διανύσματος.
2. Το άθροισμα των διαφορών υψωμένων στο τετράγωνο υπολογίζεται συχνά χωρίς να διαιρείται με το πλήθος. Χρησιμοποιείται συνήθως όταν το σφάλμα που θέλουμε να μετρήσουμε είναι αρκετά μικρό.
3. Μπορούμε να ορίσουμε επίσης την ρίζα του μέσου τετραγωνικού σφάλματος. Η συνάρτηση αυτή αντιπροσωπεύει την στατιστική διασπορά. Είναι ιδιαίτερα επιρρεπής σε ακραίες τιμές (outliers).
4. Από την επιστήμη της θεωρίας πληροφορίας η συνάρτηση cross entropy χρησιμοποιείται όλο και περισσότερο για την προσέγγιση της συνάρτησης σφάλματος του επιπέδου εξόδου.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$$

$$SSE = \sum (y - y')^2$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{j=1}^n (y_j - \hat{y}_j)^2}$$

$$\log loss = -\frac{1}{N} \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Κανονικοποίηση δεδομένων

Τα δεδομένα συνήθως δεν παρουσιάζονται στο ΤΝΔ ακριβώς στην μορφή που είναι επιθυμητή. Συνήθως τα δεδομένα κανονικοποιούνται, κλιμακώνονται δηλαδή σε ένα συγκεκριμένο εύρος σε μια διαδικασία που ονομάζεται κανονικοποίηση. Υπάρχουν αρκετοί διαφορετικοί τρόποι κανονικοποίησης των δεδομένων.

Κωδικοποίηση N κλάσεων

Ένα συχνό πρόβλημα είναι ότι οι έξοδοι που θέλουμε είναι N κατηγορίες της οποίες ονομάζουμε κλάσεις. Αυτό είναι κυρίως πρόβλημα κωδικοποίησης, αλλά συμπεριλαμβάνεται στην έννοια της κανονικοποίησης. Για να κωδικοποιηθεί ένα N σύνολο κλάσεων, πχ 3 είδη κρασιών, θα χρησιμοποιηθεί ένας νευρώνας εξόδου για κάθε κλάση του πρόβλημα(N νευρώνες). Στο παράδειγμα των κρασιών 3 νευρώνες εξόδου που ο καθένας θα αναπαριστά μια κατηγορία. Κατ' αυτόν τον τρόπο το διάνυσμα επιθυμητών τιμών κωδικοποιείται ως εξής:

$$\begin{array}{ll} 1 \rightarrow & 1, 0, 0 \\ 2 \rightarrow & 0, 1, 0 \\ 3 \rightarrow & 0, 0, 1 \end{array}$$

Αν έχετε έναν εξαιρετικά μεγάλο αριθμό κλάσεων, η κωδικοποίηση αυτή μπορεί να γίνει προβληματική επειδή πρέπει να έχετε έναν νευρώνα για κάθε κλάση. Σε τέτοιες περιπτώσεις, υπάρχουν αρκετές επιλογές. Πρώτον, ίσως βρείτε έναν τρόπο να διαλέξετε με διαφορετικό τρόπο τις κατηγορίες σας, καταλήγοντας σε μικρότερο αριθμό.

Κανονικοποίηση σε εύρος

Τα δεδομένα σπάνια βρίσκονται στο εύρος των συναρτήσεων ενεργοποίησης. Επίσης αν επιλέξουμε την αναπαράσταση N κατηγοριών σε μια διατεταγμένη λίστα από διαφορετικά εύρη, είναι απαραίτητο να φέρουμε τα δεδομένα στο εύρος που θέλουμε. Είναι επιθυμητό τα παραδείγματα εκπαίδευσης να βρίσκονται στην ίδια δυναμική περιοχή και κατά προτίμηση στο εύρος της συνάρτησης ενεργοποίησης. Αυτό σημαίνει για την εκθετική σιγμοειδή και την ReLu στο (0,1) και για την Tanh στο (-1,1).

Για την κανονικοποίηση στο διάστημα της επιλογής μας (n_{low}, n_{high}), δοθέντος ότι τα δεδομένα βρίσκονται στο εύρος (d_{low}, d_{high}) χρησιμοποιείται ο εξής τύπος :

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

Και αντίστοιχα αν είναι επιθυμητό να ανατρέσουμε την κανονικοποίηση:

$$denorm(x, d_L, d_H, n_L, n_H) = \frac{(d_L - d_H)x - (n_H \cdot d_L) + d_H \cdot n_L}{(n_L - n_H)}$$



Αν θέλουμε να φέρουμε τα δεδομένα στο διάστημα (0 , 1) αρκεί να διαιρέσουμε κάθε τιμή με την μέγιστη τιμή των μετρήσεων.

Κανονικοποίηση Z

Η πιο διαδεδομένη μέθοδος είναι η κανονικοποίηση Z(Z-norm). Η Z-norm δουλεύει είτε για έναν πραγματικό αριθμό είτε για μια ταξινομημένη λίστα N κλάσεων. Για σχεδόν όλες τις εφαρμογές, θα πρέπει να χρησιμοποιηθεί Z-norm αντί της κανονικοποίησης σε εύρος. Η Z-norm παρέχει περισσότερες πληροφορίες, τα κανονικοποιημένα δεδομένα παίρνουν θετικές και αρνητικές τιμές και βρίσκονται στο ίδια δυναμική περιοχή. Για ένα διάνυσμα μετρήσεων μιας τιμής, πχ θερμοκρασία, αν x η μετρούμενη τιμή και μ, σ η στατιστική μέση τιμή και διασπορά των μετρήσεων τότε η Z-norm τιμή υπολογίζεται ως εξής :

$$z = \frac{x - \mu}{\sigma}$$

Αυτές οι τρεις μέθοδοι καλύπτουν το μεγαλύτερο μέρος των κανονικοποιήσεων για τα περισσότερα προβλήματα σε TNΔ.



Ο αλγόριθμος οπισθοδρομικής διάδοσης του σφάλματος (BPA)

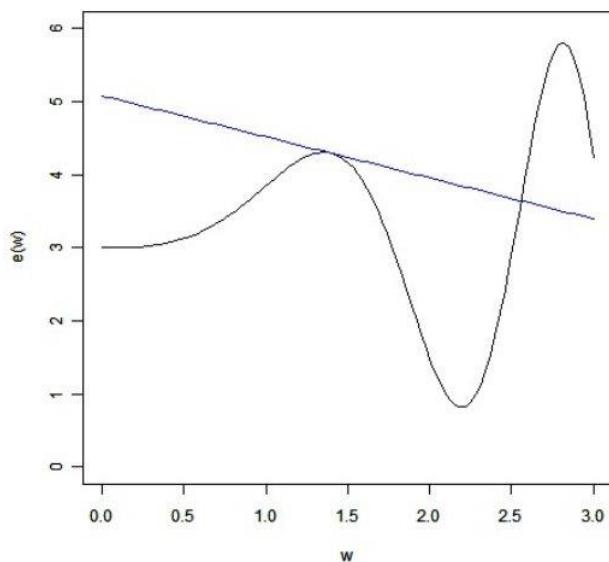
Ο αλγόριθμος οπισθοδρομικής διάδοσης του σφάλματος είναι μια από τις πιο κοινές μεθόδους εκπαίδευσης. Ανακαλύφθηκε από τούς Rumelhart, Hinton, & Williams το 1986 και παραμένει ακόμη ιδιαίτερα δημοφιλής.

Οι προγραμματιστές συχνά εκπαιδεύουν ΤΝΔ μικρού και μεγάλου βάθους με τον BPA σε GPU, επειδή κλιμακώνεται πολύ άριστα στο μοντέλο σε κάρτες γραφικής επεξεργασίας (GPU). Για να κατανοηθεί ο αλγόριθμος, πρέπει να εξετασθεί πώς επιτυγχάνεται αναπροσαρμογή των βαρών μέσα από την επεξεργασία ενός προτύπου-παραδείγματος.

Ο BPA βασίζεται στο υπολογισμό της παραγώγου για την μεταβολή των συντελεστών βαρύτητας. Για αυτό χρησιμοποιείται συχνά η ορολογία απότομης καθόδου(gradient descent), που είναι άμεσα συνδεδεμένη με τον BPA και θα αναλυθεί σε βάθος.

Gradient descent

Η gradient descent αναφέρεται στον υπολογισμό της κλίσης κάθε βάρους στο ΤΝΔ για κάθε στοιχείο εκπαίδευσης. Η κλίση(gradient) κάθε βάρους μας δίνει μια μετρική στο πώς πρέπει να αλλάξει ώστε να συγκλίνει προς την επιθυμητή τιμή. Η κλίση είναι η παράγωγος της συνάρτησης σφάλματος για το συγκεκριμένο βάρος. Με την gradient descent, υπολογίζεται η τιμή της κλίσης κάθε βάρους για να μεταβληθεί η τιμή τους προς την διεύθυνση που υποδεικνύει η παράγωγος. Ο σκοπός είναι η ελαχιστοποίηση της συνάρτησης σφάλματος.



Ο υπολογισμός της κλίσης της συνάρτησης σφάλματος επιτρέπει τον προσδιορισμό αλλαγής του βάρους. Με τη σειρά του, ο προσδιορισμός αυτός θα μειώσει το σφάλμα του δικτύου. Ο υπολογισμός της συνάρτησης σφάλματος έχει αναλυθεί στην προηγούμενη ενότητα [[link](#)]. Το σφάλμα είναι η διαφορά μεταξύ της αναμενόμενης εξόδου και της πραγματικής εξόδου του νευρικού δικτύου που μας δίνει τις ακόλουθες πληροφορίες:

Μηδενική κλίση - Το βάρος δεν αυξάνει το σφάλμα του δικτύου, άρα δεν αλλάζει.

Αρνητική κλίση - Το βάρος πρέπει να αυξηθεί για να επιτευχθεί χαμηλότερο σφάλμα.



Θετική κλίση - Το βάρος πρέπει να μειωθεί για να επιτευχθεί χαμηλότερο σφάλμα.

Η ανάλυση πραγματώνεται για δευτέρου βαθμού συνάρτηση σφάλματος(τετραγωνική) , όπως η MSE και η SSE error functions.

Υπολογισμός Δέλτα νευρώνων εξόδου

Διθέντος ότι έχουν γίνει οι υπολογισμοί εμπρόσθιας διάδοσης (χρησιμοποιώντας την συνάρτηση ενεργοποίησης φ_i με παράγωγο φ_i) γνωρίζουμε τις εξόδους του δικτύου y_i και τις επιθυμητές εξόδους ŷ_i , υπολογίζουμε τα δέλτα για κάθε νευρώνα εξόδου, σύμφωνα με την εξίσωση

$$\delta_i = (\hat{y}_i - y_i) \phi'_i$$

Αυτή η σχέση ισχύει για τετραγωνική συνάρτηση σφάλματος. Τα δέλτα κάθε νευρώνα υπολογίζονται από την διαφορά της εξόδου με την ιδεατή τιμή πολλαπλασιασμένη με την παράγωγο της συνάρτησης σφάλματος στο σημείο εκείνο.

Υπολογισμός Δέλτα κρυφών νευρώνων

Ο υπολογισμός των συντελεστών δέλτα για του κρυφούς νευρώνες εμπεριέχει όλη την πληροφορία του σφάλματος του επόμενου επιπέδου, δηλαδή τους συντελεστές δ_k και τα βάρη w_k . Σύμφωνα με αυτά οι τιμές του τρέχοντος κρυφού επιπέδου υπολογίζονται από την σχέση:

$$\delta_i = \phi'_i \sum_k w_{ki} \delta_k$$

Επειδή ο υπολογισμός των συντελεστών δ είναι αναδρομικός, από νευρώνες υψηλότερων επιπέδων προς χαμηλότερα επίπεδα , η διαδικασία επαναπροσδιορισμού των συντελεστών ακολουθεί τα εξής βήματα :

1. Αρχικοποίηση συντελεστών βαρύτητας και ρυθμού εκπαίδευσης
2. Ενεργοποίηση του δικτύου τοποθετώντας για είσοδο ένα τυχαίο πρότυπο από το σετ εκπαίδευσης υπολογίζοντας την έξοδο από το επίπεδο εισόδου προς το εξόδου.
3. Υπολογισμός των συντελεστών βαρύτητας δ, σύμφωνα με τις επιθυμητές τιμές εξόδου.
4. Επαναπροσδιορισμός των συντελεστών βαρύτητας σύμφωνα με την σχέση:

$$\frac{dE}{dw(t)} = \delta_i \cdot \varphi_j, \text{όπου } j \text{ δείκτης των συνάψεων στον νευρώνα } i \text{ και } \eta \text{ μεταβολή}$$

για ρυθμό μάθησης ε θα είναι :

$$\Delta w_{(t)} = -\epsilon \frac{\partial E}{\partial w_{(t)}}$$

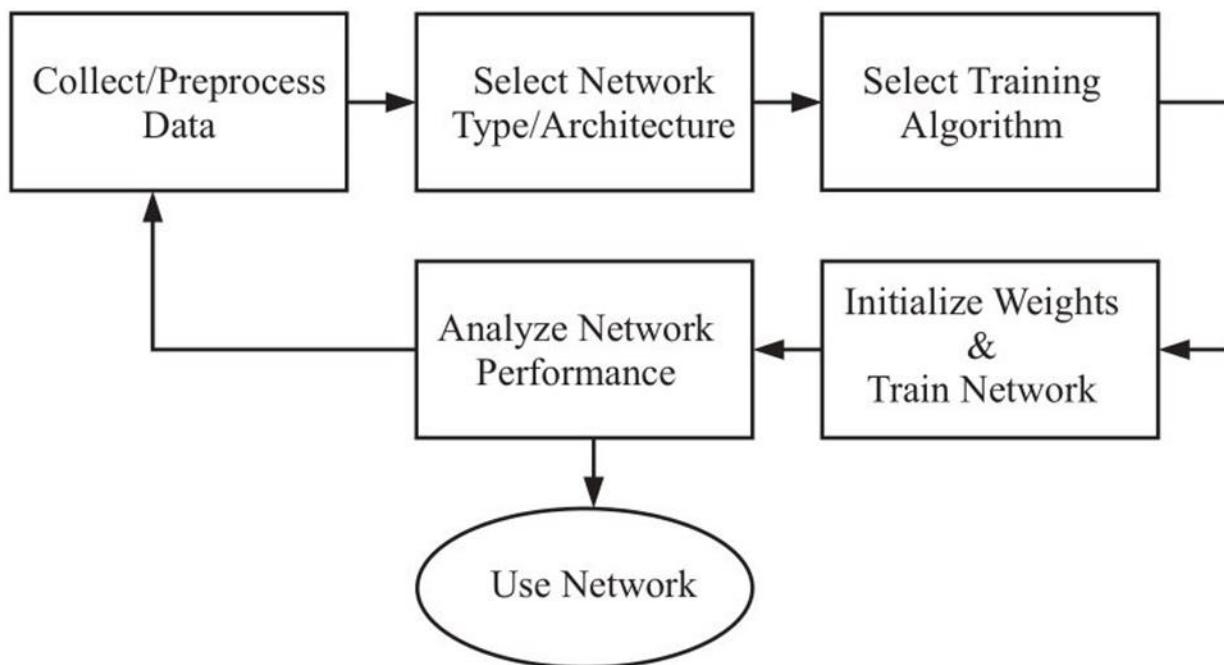


5. Έλεγχος σύγκλισης. Έχουν αναπτυχθεί διάφοροι αλγόριθμοι που εξετάζουν την σύγκλιση είτε ελέγχοντας την συνολική απόλυτη μεταβολή των συντελεστών βαρύτητας, είτε βρίσκοντας το μέσο ρυθμό μεταβολή σφάλματος είτε από τον μέσο ρυθμό μεταβολής των δ.
6. Τέλος, εφόσον δεν υπάρχει σύγκλιση ο αλγόριθμος επαναλαμβάνεται, τοποθετώντας ένα νέο παράδειγμα στην είσοδο και επαναλαμβάνοντας την ίδια διαδικασία.

Αρχιτεκτονικές σχεδιασμού

Οι θεωρητικές και πρακτικές πτυχές δεν αλληλοαποκλείονται. Μόνο συνδυάζοντας μια βαθιά γνώση των βασικών στοιχείων του δικτύου με την πρακτική εμπειρία στη χρήση νευρωνικών δικτύων, μπορούμε να αξιοποιήσουμε στο έπακρο αυτήν την τεχνολογία. Πρόκειται για μια επαναληπτική διαδικασία που αρχίζει με τη συλλογή δεδομένων και την προεπεξεργασία τους για να καταστήσει την εκπαίδευση αποτελεσματική. Σε αυτό το στάδιο, τα δεδομένα πρέπει επίσης να χωριστούν σε σύνολα εκπαίδευσης και δοκιμών.

Μετά την επιλογή των δεδομένων, πρέπει να επιλεγεί ο κατάλληλος τύπος δικτύου και η αρχιτεκτονική (π.χ. αριθμός επιπέδων, αριθμός νευρώνων). Στη συνέχεια επιλέγουμε έναν αλγόριθμο εκπαίδευσης που είναι κατάλληλος για το πρόβλημα που προσπαθούμε να επιλύσουμε. Αφού εκπαιδευτεί το δίκτυο, θέλουμε να αναλύσουμε τις επιδόσεις του. Αυτή η μελέτη μπορεί να μας οδηγήσει στην ανακάλυψη προβλημάτων με τα δεδομένα, την αρχιτεκτονική ή τον αλγόριθμο εκπαίδευσης. Ολόκληρη η διαδικασία στη συνέχεια επαναλαμβάνεται έως ότου η απόδοση είναι ικανοποιητική.

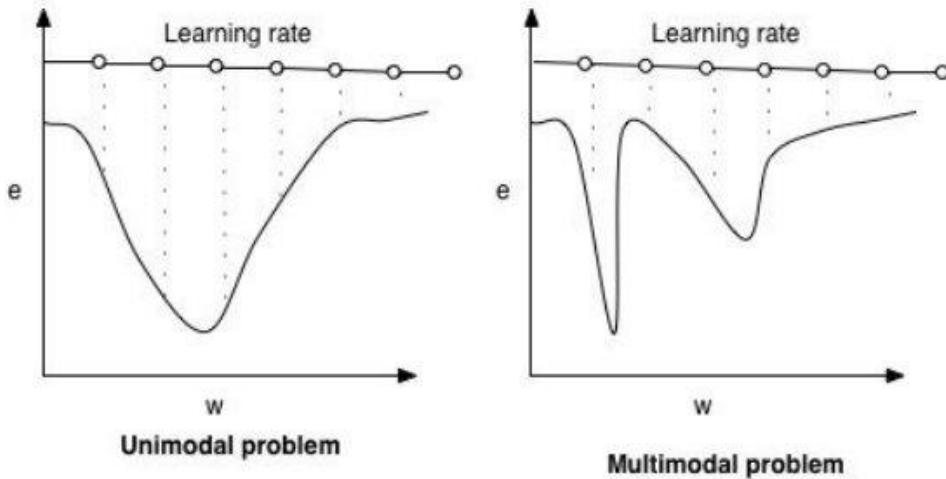


Flowchart of Neural Network Training Process



Επιλογή συντελεστή εκπαίδευσης (ϵ)

Η λανθασμένη επιλογή του συντελεστή εκπαίδευσης μπορεί να έχει 2 αποτελέσματα. Αν ο συντελεστής έχει μεγάλη αριθμητική τιμή δεν είναι εγγυημένη η σύγκλιση σε τοπικό ελάχιστο αλλά να παρατηρούνται ταλαντώσεις γύρω από τουλάχιστον ένα τοπικό ελάχιστο. Οι ταλαντώσεις είναι άμεσα εξαρτώμενες από την τιμή του ρυθμού μάθησης. Αντίθετα, η επιλογή μικρής αριθμητικής τιμής η σύγκλιση μπορεί να απαιτεί τεράστιο αριθμό επαναλήψεων, ενώ αυξάνεται η πιθανότητα σύγκλισης σε κακό τοπικό ελάχιστο, αν υπάρχει.



Τα παραπάνω δύο γραφήματα δείχνουν τη σχέση μεταξύ βάρους και σφάλματος ενός ΤΝΔ. Καθώς το πρόγραμμα αυξάνει ή μειώνει ένα μεμονωμένο βάρος, το σφάλμα αλλάζει. Ένα πρόβλημα με ένα ελάχιστο(unimodal) είναι συνήθως εύκολο να λυθεί επειδή το γράφημά του έχει μια βέλτιστη λύση. Σε αυτή την περίπτωση, θεωρούμε καλή βαθμολογία να είναι χαμηλό ποσοστό σφάλματος. Ένα πολυτροπικό(multimodal) πρόβλημα έχει πολλές ανωμαλίες ή πιθανές καλές λύσεις. Εάν το πρόβλημα είναι απλό (unimodal), ένα γρήγορος ρυθμός εκμάθησης είναι η καλύτερη επιλογή. Ωστόσο, η επιλογή μεγάλου συντελεστή μάθησης κάνει το δεύτερο διάγραμμα να αποτυγχάνει να βρει την επιθυμητή λύση(τα δύο ακρότατα).

Μελέτη αλγορίθμου οπισθοδιάδοσης(BPA)

Ο αλγόριθμος οπισθοδρομικής διάδοσης (BPA) εξασφαλίζει την σύγκλιση σε τοπικό ελάχιστο, δηλαδή για μικρές μεταβολές των βαρών το σφάλμα δεν θα μειωθεί. Οι παρακάτω οδηγίες που έχουν ληφθεί κυρίως από δοκιμές μας βοηθούν στην αρχιτεκτονική κατασκευής ενός μοντέλου νευρώνων.

1. Τα παραδείγματα του σετ εκπαίδευσης πρέπει να επιλέγονται με τυχαίο τρόπο και περισσότερες από μια φορά το κάθε ένα.
2. Ο αριθμός των νευρώνων που αποτελείται κάθε επίπεδο του ΤΝΔ πρέπει να μειώνεται από την είσοδο προς την έξοδο. Με αυτή την αρχιτεκτονική είναι πιο πιθανό να αποφευχθούν τοπικά ελάχιστα.
3. Η τιμή του ρυθμού εκπαίδευσης επιλέγεται εμπειρικά με δοκιμές.
4. Επειδή οι συντελεστές δ των χαμηλότερων επιπέδων έχουν μικρότερες τιμές, επιλέγεται μεγαλύτερος συντελεστής εκπαίδευσης.



5. Η σύγκλιση επιβραδύνεται σημαντικά με την αύξηση των επιπέδων, γι' αυτό χρησιμοποιούνται όσα λιγότερα γίνονται. Προσπαθούμε να προσθέτουμε νευρώνες στα ήδη υπάρχοντα επίπεδα και εν συνεχεία εξετάζεται η προσθήκη επιπέδου.
6. Οι επιθυμητές τιμές πρέπει να ανήκουν στο πεδίο τιμών της συνάρτησης ενεργοποίησης. Πιο συγκεκριμένα αν η συνάρτηση περιορίζει το πεδίο τιμών στο διάστημα $[-1, 1]$ οι επιθυμητές τιμές να ανήκουν στο διάστημα $[-1+\varepsilon_1, +1-\varepsilon_2]$, όπου $\varepsilon_1, \varepsilon_2$ μικρές θετικές τιμές.
7. Με το bias νευρώνα αλλά και με άλλες τεχνικές επιδιώκεται η επέκταση του διανύσματος εισόδου με σκοπό να αυξηθούν οι βαθμοί ελευθερίας, ελαχιστοποιώντας την πιθανότητα τοπικού ελαχίστου.

Overfitting and Underfitting

Η μεγαλύτερη πρόκληση στη μηχανική μάθηση(ML) είναι ότι πρέπει το μοντέλο να σημειώσει καλό ποσοστό επιτυχίας σε νέες, άγνωστες εισόδους - όχι μόνο σε εκείνες που εκπαιδεύτηκε. Η ικανότητα να αποδίδει ικανοποιητικά αποτελέσματα σε καινούργιες εισόδους ονομάζεται **γενίκευση(generalization)**. Συνήθως, όταν εκπαιδεύουμε ένα μοντέλο μηχανικής μάθησης, έχουμε πρόσβαση σε ένα πεπερασμένο σετ δεδομένων εκπαίδευσης. Κατά την διάρκεια της εκπαίδευσης μπορεί να υπολογιστεί κάποιο μέτρο σφάλματος στο σετ (σφάλμα εκπαίδευσης), το οποίο στοχεύουμε να μειώσουμε. Μέχρι στιγμής, αυτό που περιγράφηκε είναι απλώς ένα πρόβλημα βελτιστοποίησης. Αυτό που χωρίζει τη μηχανική μάθηση από τη βελτιστοποίηση είναι ότι θέλουμε να ελαχιστοποιήσουμε το σφάλμα γενίκευσης, που ονομάζεται επίσης σφάλμα δοκιμής. Το σφάλμα γενίκευσης ορίζεται ως η αναμενόμενη τιμή του σφάλματος σε μια νέα είσοδο. Εδώ η απαιτήσεις που αναμένονται λαμβάνονται σε διάφορες πιθανές εισόδους, που προκύπτουν από τη κατανομή των εισόδων που αναμένουμε ότι το σύστημα θα συναντήσει στην πράξη.

Εκτιμούμε συνήθως το σφάλμα γενίκευσης ενός μοντέλου μηχανικής μάθησης, μετρώντας την απόδοσή του σε ένα σετ δοκιμών (test set) παραδειγμάτων που συλλέχθηκαν χωριστά από το σετ εκπαίδευσης. Πώς όμως μπορούμε να επηρεάσουμε την απόδοση στο σετ δοκιμών όταν παρατηρούμε μόνο το σύνολο εκπαίδευσης;

Το πεδίο της στατιστικής θεωρίας μάθησης παρέχει μερικές απαντήσεις. Εάν η εκπαίδευση και το σετ δοκιμών συλλέγονται αυθαίρετα, υπάρχουν πράγματα λίγα πράγματα που μπορούμε να κάνουμε. Αν έχουμε τη δυνατότητα να κάνουμε κάποιες υποθέσεις σχετικά με το πώς συλλέγονται το σύνολο εκπαίδευσης και δοκιμών, τότε μπορούμε να εξάγουμε ορισμένα συμπεράσματα.

Συνήθως κάνουμε μια σειρά από υποθέσεις όπως ότι τα παραδείγματα σε κάθε σύνολο δεδομένων είναι ανεξάρτητα το ένα από το άλλο και ότι το σετ εκπαίδευσης και το σετ δοκιμών κατανέμονται πανομοιότυπα, αντλούνται δηλαδή από την ίδια κατανομή πιθανότητας. Αυτή η υπόθεση μας επιτρέπει να περιγράψουμε τη διαδικασία παραγωγής δεδομένων με κατανομή πιθανότητας σε ένα μόνο παράδειγμα. Στηριζόμενοι σε αυτό το πιθανοτικό πλαίσιο, καταφέρνουμε να μελετήσουμε μαθηματικά τη σχέση μεταξύ σφάλματος κατάρτισης και σφάλματος δοκιμής.

Ας υποθέσουμε ότι έχουμε μια κατανομή πιθανότητας $p(x, y)$ και δειγματοληπτούμε επανειλημμένα για τη δημιουργία των δύο συνόλων. Για κάποια σταθερή τιμή w , το αναμενόμενο σφάλμα είναι ακριβώς το ίδιο και στα δύο σετ, επειδή και οι δύο αναμενόμενες τιμές υπολογίζονται



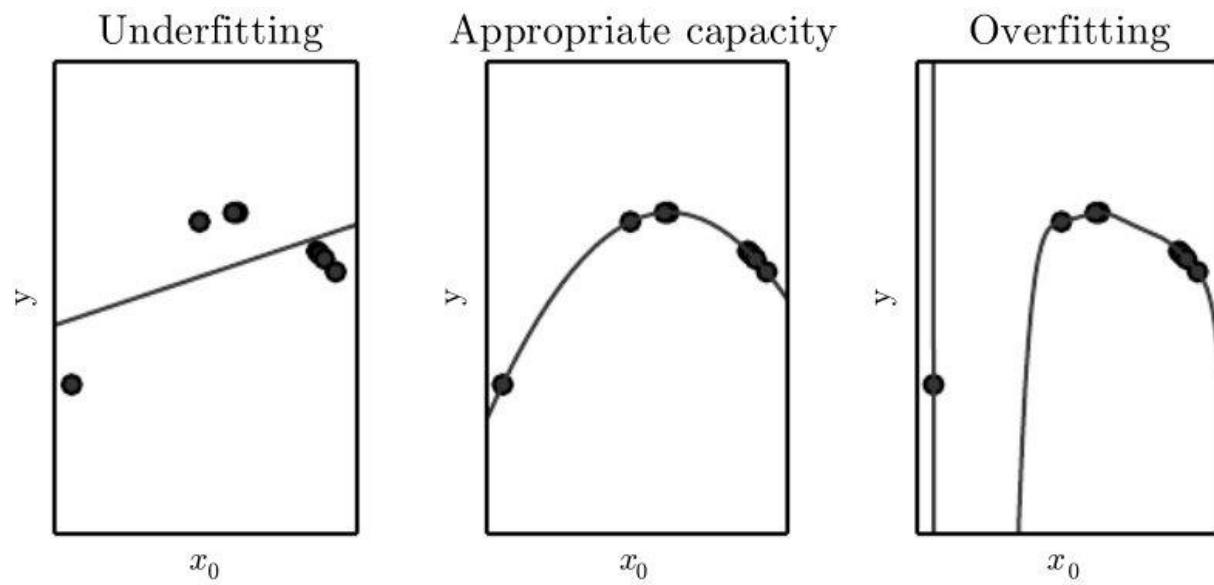
χρησιμοποιώντας την ίδια διαδικασία δειγματοληψίας του συνόλου δεδομένων. Η μόνη διαφορά μεταξύ των δύο συνθηκών είναι το όνομα που αποδίδουμε στα σύνολα δεδομένων.

Η διαδικασία στην επιστήμη της μηχανικής μάθησης που ακολουθείται είναι η εξής: δειγματοληπτούμε το σετ εκπαίδευσης και στη συνέχεια το χρησιμοποιούμε για να επιλέξουμε τις παραμέτρους του μοντέλου για να μειώσουμε το σφάλμα. Στη συνέχεια δοκιμάσουμε το εκπαιδευμένο σύστημα στο σετ δοκιμών. Σύμφωνα με αυτή τη διαδικασία, το αναμενόμενο σφάλμα δοκιμής είναι μεγαλύτερο ή ίσο με την αναμενόμενη τιμή του σφάλματος εκπαίδευσης.

Οι παράγοντες που καθορίζουν πόσο καλά αποδίδει ένας αλγόριθμος μηχανικής μάθησης είναι η ικανότητά του να:

1. Κάνει το σφάλμα εκπαίδευσης αρκετά μικρό. (όχι ελάχιστο)
2. Κάνει το χάσμα(gap) ανάμεσα στο σφάλμα εκπαίδευσης και δοκιμής μικρό.

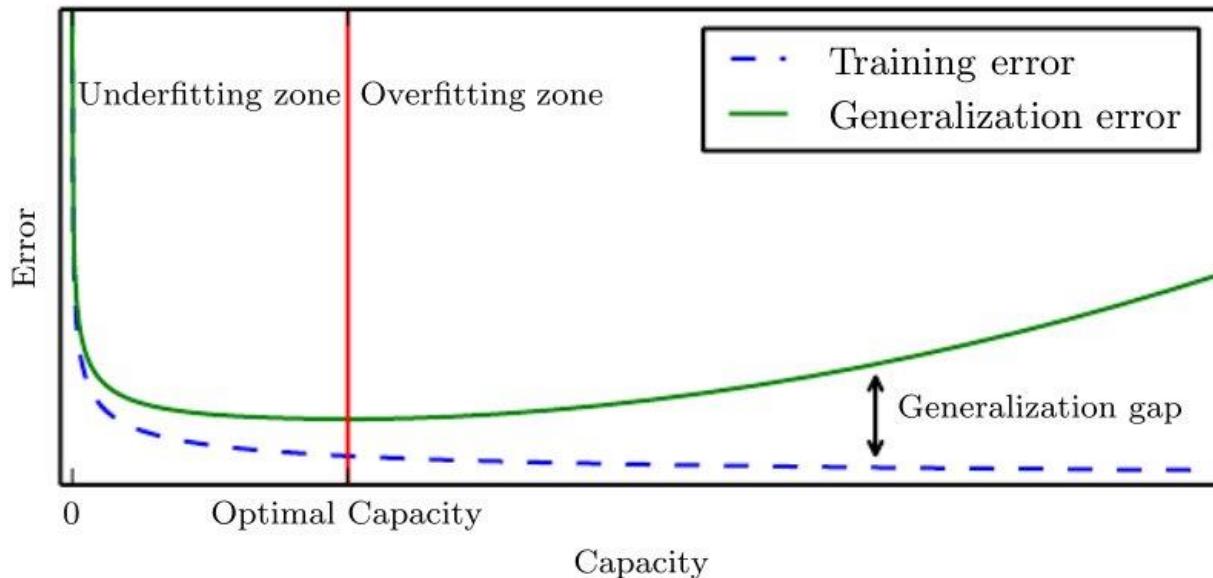
Αυτοί οι δύο παράγοντες αντιστοιχούν στις δύο βασικές προκλήσεις της μηχανικής μάθησης: **underfitting** και **overfitting**. Το **underfitting** συμβαίνει όταν το μοντέλο δεν είναι σε θέση να επιτύχει επαρκώς χαμηλή τιμή σφάλματος στο σετ εκπαίδευσης. Το **overfitting**(υπερεκπαίδευση) συμβαίνει όταν το χάσμα ανάμεσα στο σφάλμα εκπαίδευσης και δοκιμής είναι αρκετά μεγάλο. Με το παρακάτω παράδειγμα αυτά που αναφέρθηκαν γίνονται πλήρως κατανοητά.



Τοποθετούνται τρία διαφορετικά μοντέλα στο σετ εκπαίδευσης. Τα δεδομένα εκπαίδευσης δημιουργήθηκαν δειγματοληπτόντας τις x και y τιμές. (Αριστερά) Μια γραμμική συνάρτηση που ταιριάζει με τα δεδομένα πάσχει από underfitting - δεν μπορεί να καταγράψει την καμπυλότητα στα δεδομένα. Στο κέντρο, ένα τετραγωνικό πολυώνυμο ταιριάζει καλά τα δεδομένα, δηλαδή δημιουργεί γενίκευση σε αόρατα σημεία. Στα δεξιά, ένα πολυώνυμο ενάτου βαθμού ταιριάζει στα δεδομένα αλλά πάσχει από overfitting, δεν μπορεί να αναπαραστήσει επαρκώς άλλα δεδομένα. Η λύση περνά ακριβώς από όλα τα σημεία εκπαίδευσης, αλλά δεν εξάγουμε τη σωστή δομή. Υπάρχει μια βαθιά κοιλάδα ανάμεσα σε δύο σημεία εκπαίδευσης που δεν εμφανίζονται στην πραγματική υποκείμενη συνάρτηση. Επίσης αυξάνεται απότομα στην αριστερή πλευρά των δεδομένων, ενώ η πραγματική



λειτουργία μειώνεται στην περιοχή αυτή. Ορίζοντας ως ικανότητα ανταπόκρισης(representational capacity) τον καθορισμό της ομάδας των συναρτήσεων ενός αλγορίθμου μάθησης κατά τη μεταβολή των παραμέτρων προκειμένου να μειωθεί ο στόχος της εκπαίδευσης. Συνοψίζονται όσα αναφέρθηκαν στο διάγραμμα που ακολουθεί:



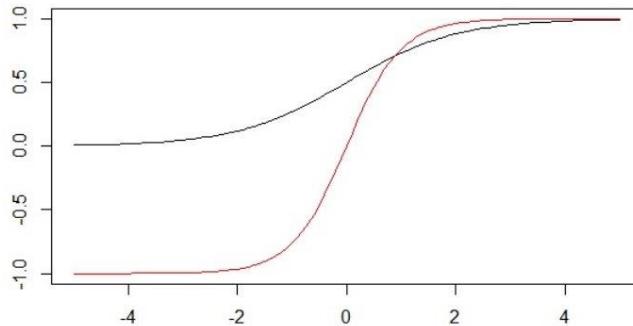
Παρουσιάζεται μια τυπική σχέση μεταξύ ικανότητα ανταπόκρισης(capacity) και σφάλματος. **Το σφάλμα εκπαίδευσης και δοκιμής συμπεριφέρονται διαφορετικά.** Αριστερά του γραφήματος, το σφάλμα κατάρτισης και το σφάλμα γενίκευσης είναι και τα δύο υψηλά. Αυτό σπάνια υφίσταται. Καθώς αυξάνουμε την ικανότητα ανταπόκρισης, το σφάλμα εκπαίδευσης μειώνεται, αλλά το χάσμα μεταξύ των σφαλμάτων αυξάνεται. Όταν το μέγεθος αυτού του χάσματος ξεπερνά τη μείωση του σφάλματος εκπαίδευσης, το μοντέλο εισέρχεται στην περιοχή της υπερεκπαίδευσης.

Τα TNΔ δεν διαφέρουν στην συμπεριφορά τους από τα άλλα μοντέλα μηχανικής μάθησης. Κατά την σχεδίαση οφείλουμε να προσδιορίζουμε, ανάλογα τα δεδομένα εκπαίδευσης, της παραμέτρους του δικτύου. Στην γενική περίπτωση **ο αριθμός των παραμέτρων επιλέγεται να είναι μικρότερος των δεδομένων εκπαίδευσης, τουλάχιστον κατά μια τάξη μεγέθους.** Επίσης οι κατάλληλες κανονικοποιήσεις βοηθάνε στην αποφυγή της υπερεκπαίδευσης.

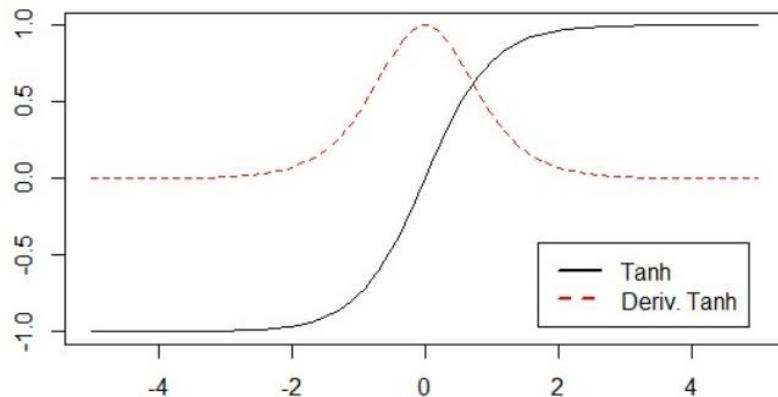
Επιλογή συνάρτησης ενεργοποίησης και σφάλματος

Όσον αφορά την επιλογή συνάρτησης ενεργοποίησης ενδείκνυται η επιλογή της tanh αντί της εκθετικής σιγμοειδής επειδή έχει μεγαλύτερο πεδίο τιμών.





Όπως μπορείτε να δείτε από το σχήμα, η υπερβολική εφαπτομένη εκτείνεται σε πολύ μεγαλύτερο εύρος από το εκθετική. Η κανονικοποίηση των δεδομένων εισόδου γίνεται στο [-1 , 1] και για τις δύο συναρτήσεις. O Kalman & Kwasny (1992) [\[link\]](#) συνιστούν υπερβολική εφαπτομένη σε όλες τις καταστάσεις αντί για την σιγμοειδή. Αυτή η σύσταση αντιστοιχεί σε πολλές πηγές της βιβλιογραφίας. Ωστόσο, αυτό το επιχείρημα εκτείνεται μόνο στην επιλογή μεταξύ της κατηγορίας των σιγμοειδών συναρτήσεων ενεργοποίησης. Βέβαια, και οι δύο υποφέρουν από το φαινόμενο κορεσμού. Κατά τον υπολογισμό της παραγώγου στα άκρα του πεδίου τιμών η παράγωγος μηδενίζεται περιορίζοντας τις δυνατότητες του αλγορίθμου, όπως φαίνεται στο διάγραμμα.



Το φαινόμενο κορεσμού των σιγμοειδών συναρτήσεων ενεργοποίησης οδηγεί τα βάρη σε ακραίες τιμές όταν ο αριθμός των επαναλήψεων είναι μεγάλος. Αυτό έχει ως αποτέλεσμα να ταξινομούν τα παραδείγματα στην κατηγορία που ανήκουν τα περισσότερα παραδείγματα του σετ εκπαίδευσης. Το τελευταίο αντιμετωπίζεται χρησιμοποιώντας την ReLU συνάρτηση για τα κρυφά επίπεδα, ιδιαίτερα σε προβλήματα που είναι έντονα μη γραμμικά. Ένα αυξανόμενο κοινό από πρόσφατες έρευνες προτείνει την ReLU σε όλες τις περιπτώσεις.

Η τετραγωνική συνάρτηση σφάλματος(MSE) μπορεί μερικές φορές να απαιτήσει πολύ χρόνο για την σωστή προσαρμογή του βάρους. Ενδείκνυται, σε ορισμένες περιπτώσεις ο υπολογισμός σφάλματος από την συνάρτηση εντροπίας(cross entropy ή αλλιώς log loss) που δίνεται από τον τύπο:



$$\text{log loss} = -\frac{1}{N} \sum_{i=1}^N (y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i))$$

Ο υπολογισμός των δέλτα για τους νευρώνες εξόδου, σύμφωνα με τον κανόνα της αλυσίδας θα γίνει:

$$\delta_i = \hat{y}_i - y_i$$

Η επιλογή αυτής της συνάρτησης μπορεί να δώσει καλύτερες προσεγγίσεις του σφάλματος και να δώσει καλύτερες λύσεις. Παρόλα αυτά θα πρέπει να δοκιμαστεί στο συγκεκριμένο πρόβλημα για ασφαλή συμπεράσματα.

Ανά παρτίδα διόρθωση βαρών(Batch size training)

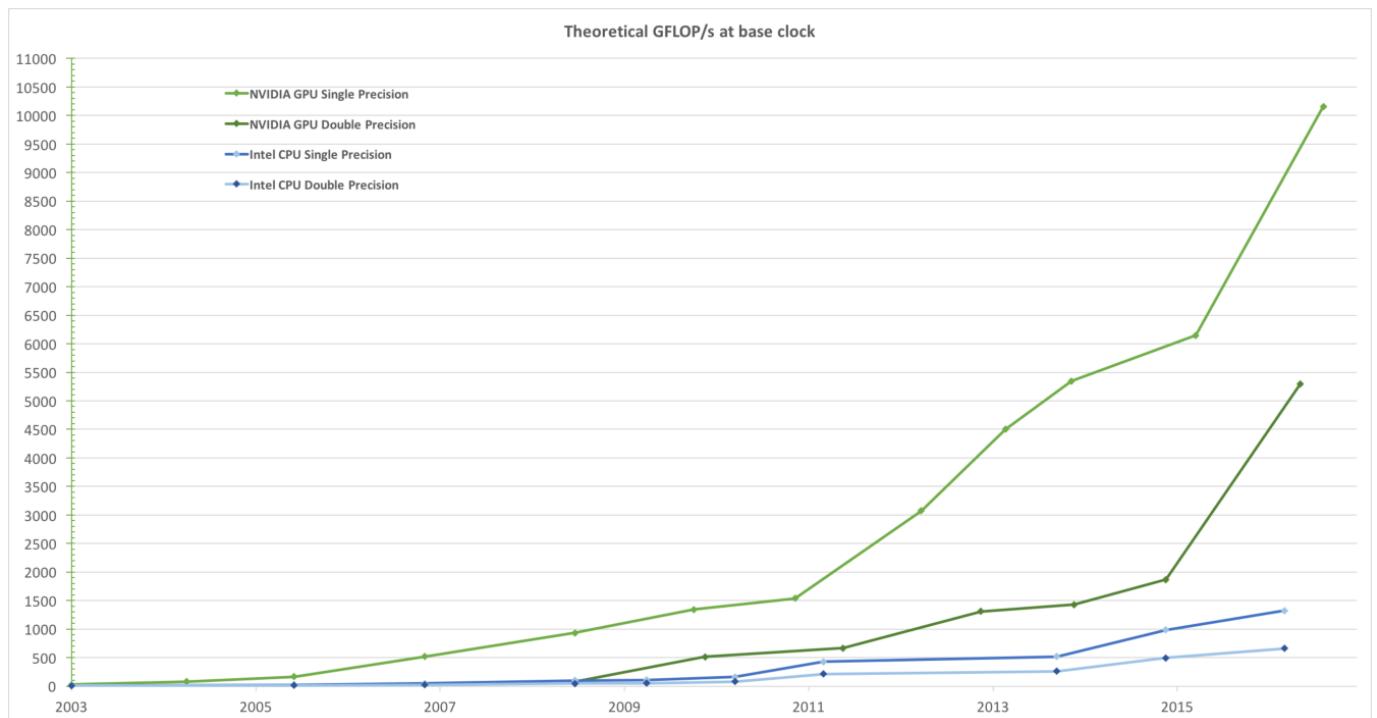
Μια εναλλακτική είναι ο αριθμός των παραδειγμάτων που θα αναπροσαρμόσουν μια φορά τα βάρη να είναι μεγαλύτερος του ενός. Ο αριθμός αυτός ονομάζεται παρτίδα(Batch size). Το πρόγραμμα αθροίζει όλες τις gradients και τα σφάλματα για μια ενιαία παρτίδα προτού ενημερώσει τα βάρη. Ένα μέγεθος παρτίδας 1 υποδηλώνει ότι τα βάρη ενημερώνονται για κάθε στοιχείο σετ εκπαίδευσης, όπως έχει περιγραφεί ο αλγόριθμος μέχρι τώρα. Ένα καλό σημείο εκκίνησης για αυτήν την τεχνική είναι ένα μέγεθος παρτίδας ίσο με το 10% του συνόλου εκπαίδευσης. Μπορείτε να αυξήσετε ή να μειώσετε το μέγεθος της παρτίδας για να δείτε την επίδρασή της στην αποτελεσματικότητα της εκπαίδευσης. Η μείωση του μεγέθους της παρτίδας, δεν θα έχει δραστική επίδραση στο χρόνο εκτέλεσης μιας επανάληψης στον τυποποιημένο backpropagation.



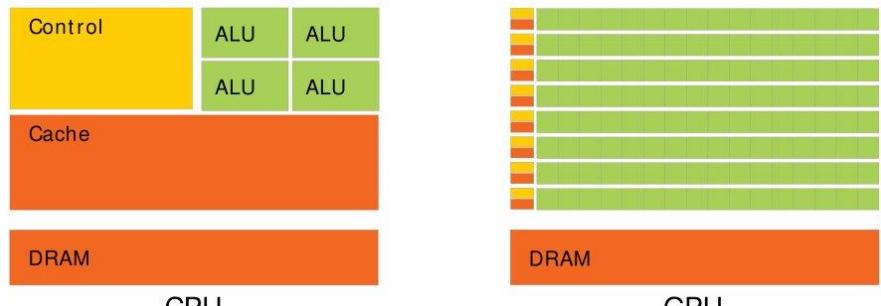
2.Παράλληλη Επεξεργασία και Cuda

Εισαγωγή

Η παράλληλη επεξεργασία είναι ένας τύπος υπολογισμού στον οποίο πολλοί υπολογισμοί ή η εκτέλεση των διαδικασιών εκτελούνται ταυτόχρονα. Μεγάλα προβλήματα συχνά μπορούν να χωριστούν σε μικρότερα, τα οποία μπορούν στη συνέχεια να λυθούν ταυτόχρονα. Εξαιτίας της απαράμιλλης ζήτησης της αγοράς για τρισδιάστατα γραφικά υψηλής ευκρίνειας σε πραγματικό χρόνο, η GPU έχει εξελιχθεί σε έναν παράλληλο επεξεργαστή, πολλών πυρήνων, με τεράστια υπολογιστική δύναμη και μεγάλο εύρος ζώνης μνήμης. Στο κάτωθι διάγραμμα φαίνεται η σύγκριση με την CPU όσον αφορά την θεωρητική δυνατότητα πράξεων.



Ο λόγος πίσω από την απόκλιση της υπολογιστικής ικανότητας μεταξύ της CPU και της GPU είναι ότι η GPU είναι εξειδικευμένη για εντατικούς παράλληλους υπολογισμούς. Έτσι σχεδιάστηκε έτσι ώστε περισσότερα τρανζίστορ να αφιερώνονται στην επεξεργασία δεδομένων και στις αριθμητικές λογικές πράξεις αντί για προσωρινή αποθήκευση δεδομένων και έλεγχο ροής.



Πιο συγκεκριμένα, η GPU είναι κατάλληλη για την αντιμετώπιση προβλημάτων που εκφράζονται ως υπολογισμοί όπου το ίδιο πρόγραμμα εκτελείται σε πολλά δεδομένα παράλληλα, με υψηλή



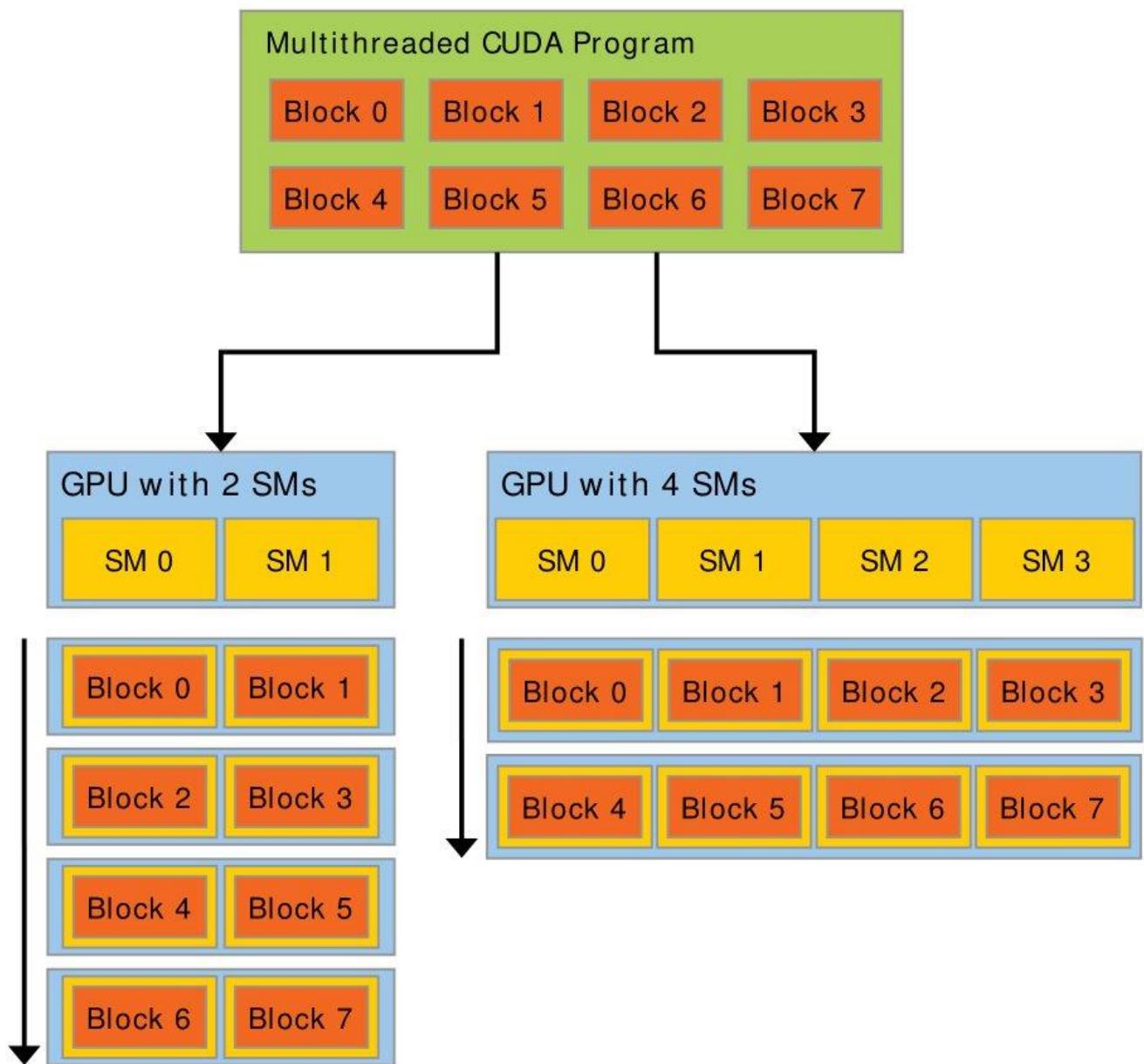
αριθμητική πυκνότητα. Επειδή το ίδιο πρόγραμμα εκτελείται για κάθε στοιχείο δεδομένων, υπάρχει μικρότερη απαίτηση για έλεγχο ροής. Πολλές εφαρμογές που επεξεργάζονται μεγάλα σύνολα δεδομένων μπορούν να χρησιμοποιήσουν ένα μοντέλο παράλληλου προγραμματισμού δεδομένων για την επιτάχυνση των υπολογισμών τους.

Το 2006 η εταιρεία Nvidia κυκλοφόρησε το CUDA, μια πλατφόρμα γενικής χρήσης για παράλληλη επεξεργασία. Εισήγαγε ένα νέο προγραμματιστικό μοντέλο που εκμεταλλεύεται μια Nvidia GPU με σκοπό την επίλυση πολλών πολύπλοκων προβλημάτων με έναν πιο αποδοτικό τρόπο από την CPU. Το CUDA έρχεται με ένα περιβάλλον λογισμικού που επιτρέπει στους προγραμματιστές να χρησιμοποιούν το C ή και άλλες γλώσσες υψηλού επιπέδου όπως δείχνεται στην εικόνα.

| GPU Computing Applications | | | | | | |
|---|--|---------------|-----------------------------|------------------------|------------------------------|-----------------------|
| Libraries and Middleware | | | | | | |
| cuFFT cuBLAS cuRAND cuSPARSE | CULA MAGMA | Thrust NPP | VSIPL SVM OpenCurrent | PhysX OptiX iRay | cuDNN TensorRT | MATLAB Mathematica |
| Programming Languages | | | | | | |
| C | C++ | Fortran | Java Python Wrappers | DirectCompute | Directives (e.g. OpenACC) | |
|  CUDA-enabled NVIDIA GPUs | | | | | | |
| Pascal Architecture (compute capabilities 6.x) | GeForce 1000 Series | | Quadro P Series | | Tesla P Series | |
| Maxwell Architecture (compute capabilities 5.x) | GeForce 900 Series | | Quadro M Series | | Tesla M Series | |
| Kepler Architecture (compute capabilities 3.x) | GeForce 700 Series GeForce 600 Series | | Quadro K Series | | Tesla K Series | |



Το μοντέλο παράλληλου προγραμματισμού CUDA έχει σχεδιαστεί για να ξεπεράσει την πρόκληση της κλιμάκωσης, διατηρώντας ταυτόχρονα μια λογική καμπύλη εκμάθησης για γνώστες της γλώσσας όπως η C. Στον πυρήνα του υπάρχουν τρεις βασικές νέες αφαιρετικές έννοιες, η ιεραρχία οργάνωσης νήματος, κοινές μνήμες και εργαλεία συγχρονισμού. Παρακάτω δείχνεται πως αποφασίζει η πλατφόρμα CUDA, με αυτόματη κλιμάκωση, να εκτελέσει το ίδιο λογισμικό σε δύο διαφορετικές κάρτες γραφικών με διαφορετικό αριθμό SM(Streaming Multiprocessor). Ένα πολυνηματικό πρόγραμμα χωρίζεται σε μπλοκ(block) νημάτων που εκτελούνται ανεξάρτητα το ένα από το άλλο, έτσι ώστε μια GPU με περισσότερους επεξεργαστές να εκτελεί αυτόματα το πρόγραμμα σε λιγότερο χρόνο από μια GPU με λιγότερους.



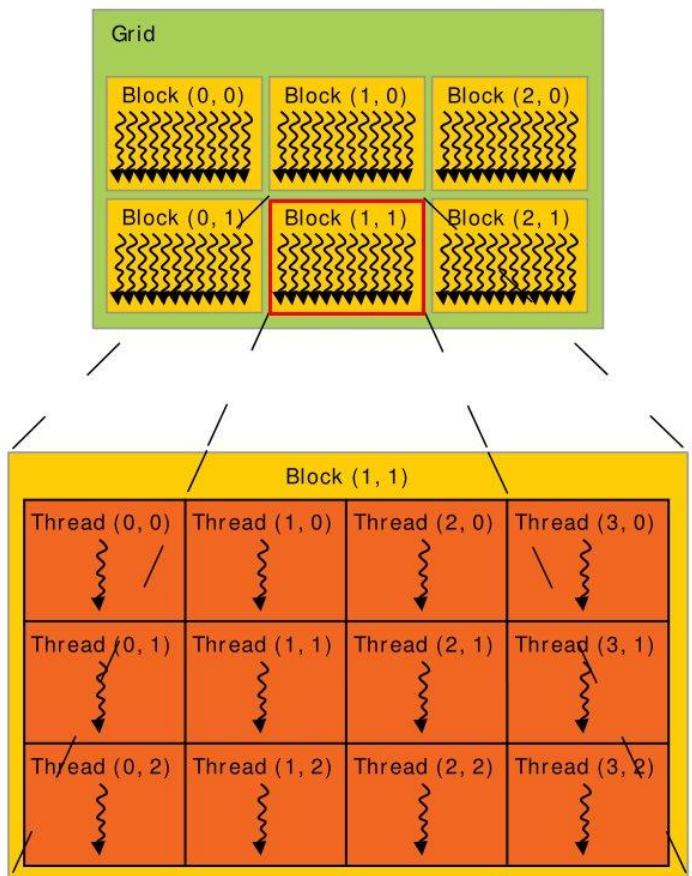
Ο πυρήνας και η ιεραρχία νημάτων

Πυρήνας(**Kernel**) είναι η συνάρτηση που εκτελείται παράλληλα στην GPU και καλείται από την CPU Ν φορές. Συναρτήσεις που καλούνται από CPU έχουν το πρόθεμα `_global_` ενώ πόσα νήματα θα εκτελεστούν στο Kernel εκφράζεται κατά την κλήση του στην CPU με το `<<< blocks of threads , threads per block >>>`. Το γνωστό παράδειγμα ενός πυρήνα πρόσθεσης 2 διανυσμάτων στα δεξιά διευκρινίζει όσα περιγράφηκαν.

Τα νήματα οργανώνονται σε **block** και **threads** ανά **block** όπως αναφέρθηκε. Για ευκολία διευθυνσηδότησης κάθε νήμα γνωρίζει που ανήκει από τα δυο τρισδιάστατα διανύσματα **threadIdx** (`threadIdx.x`, `threadIdx.y`, `threadIdx.z`) και **blockIdx**(ομοίως). Αυτό παρέχει ένα φυσικό τρόπο διευθυνσηδότησης και ανάγωγης σε προβλήματα διανυσμάτων(μονοδιάστατα), μητρώων ή όγκου(τρισδιάστατα). Υπάρχει ένα όριο στον αριθμό των νημάτων ανά μπλοκ, αφού όλα τα νήματα ενός μπλοκ αναμένεται να βρίσκονται στον ίδιο πυρήνα επεξεργαστή και πρέπει να μοιράζονται τους περιορισμένους πόρους μνήμης αυτού του πυρήνα. Στις τρέχουσες μονάδες GPU, ένα μπλοκ μπορεί να περιέχει μέχρι 1024 νήματα. Κατά αντιστοιχία εισάγεται μια αφαιρετική έννοια οργάνωσης πολλών μπλοκ από νήματα που ονομάζεται **grid**. Μια οπτική αναπαράσταση της γενικευμένης οργάνωσης των νημάτων παρουσιάζεται στο διπλανό σχήμα.

```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

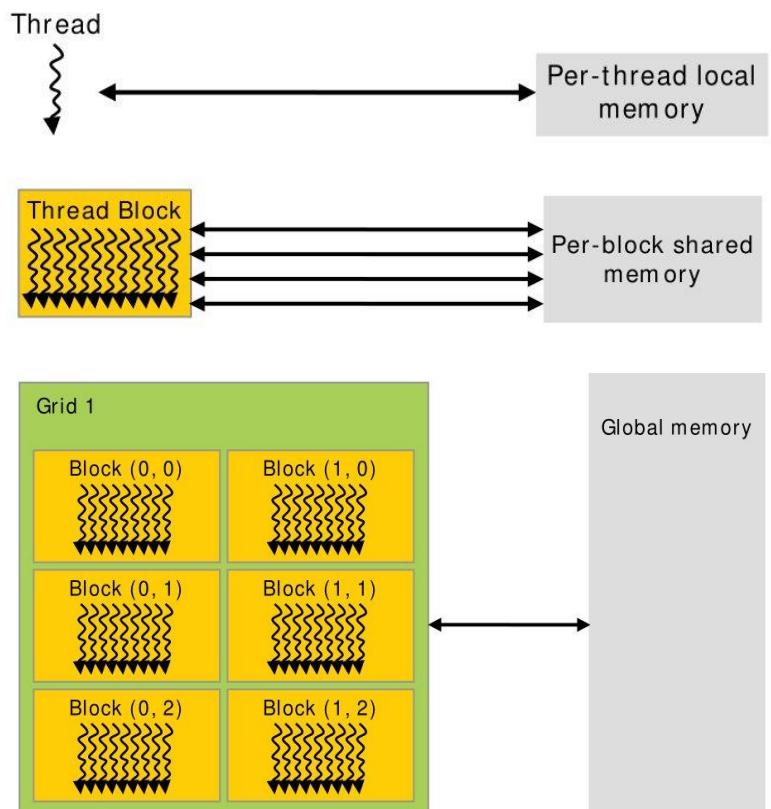
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
    ...
}
```



Ιεραρχία μνήμης

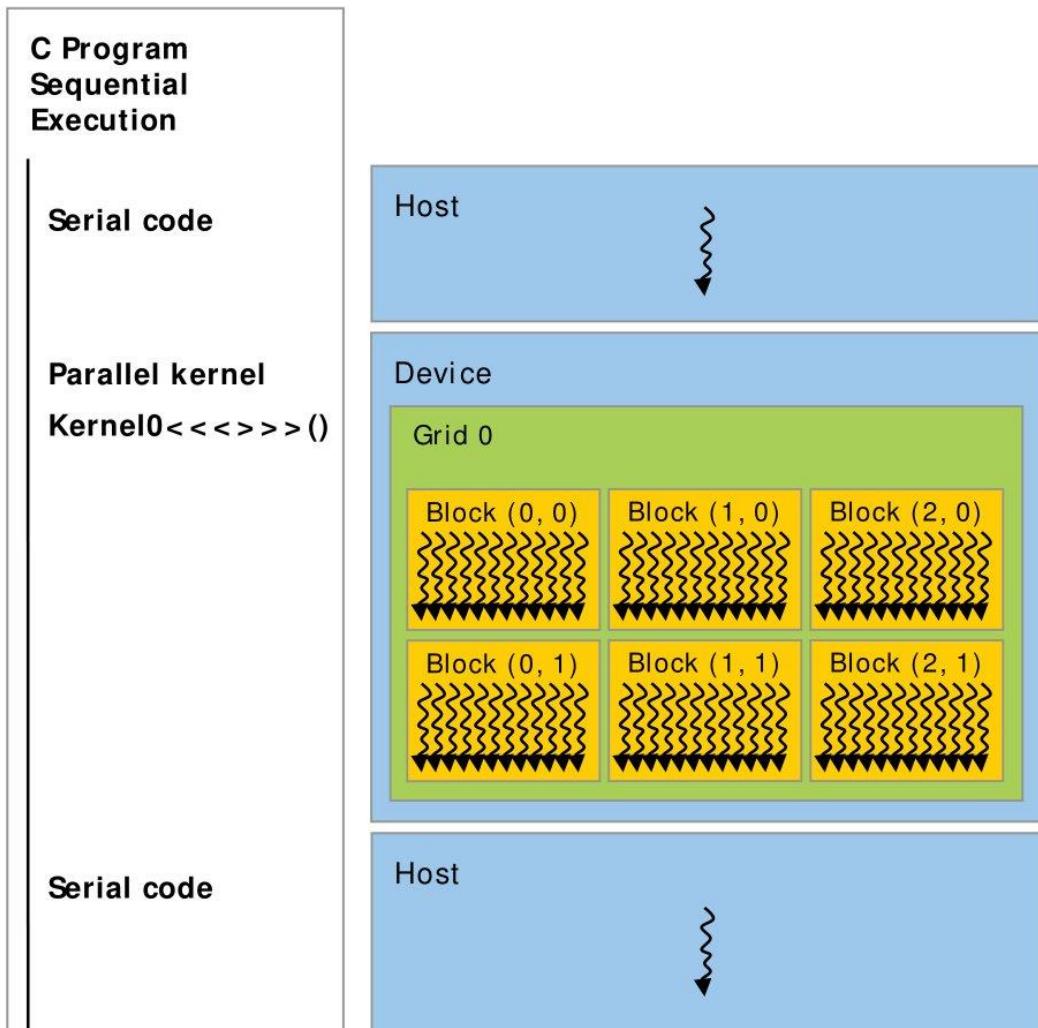
Τα νήματα μπορούν να έχουν πρόσβαση σε δεδομένα από διαφορετικούς χώρους μνήμης κατά την διάρκεια εκτέλεσης του πυρήνα(kernel). Κάθε thread έχει την δική του προσωπική(private) μνήμη. Κάθε μπλοκ από thread έχει την διαμοιραζόμενη(shared) μνήμη ορατή μόνο στα threads που ανήκουν σε αυτό το μπλοκ. Όλα τα threads έχουν πρόσβαση στην καθολική(global) μνήμη της GPU. Από πλευράς ταχύτητας μνήμης η πιο γρήγορη είναι η προσωπική μνήμη , που διαρκεί όσο εκτελείται το συγκεκριμένο νήμα. Η εν συνεχείᾳ γρηγορότερη είναι η διαμοιραζόμενη μνήμη που υπάρχει όσο εκτελούνται τα threads ενός μπλοκ. Η πιο αργή είναι η καθολική μνήμη που ο χρόνος που απαιτείται για πρόσβαση στα δεδομένα μπορεί να μειώσει σημαντικά την συνολική επιτάχυνση. Η διαμοιραζόμενη(ή αλλιώς κοινή) μνήμη αναμένεται να είναι πολύ ταχύτερη από την καθολική μνήμη.

Επομένως οποιαδήποτε ευκαιρία για την αντικατάσταση προσπελάσεων της καθολικής μνήμης με πρόσβαση στην κοινή μνήμη θα πρέπει να εκμεταλλεύεται.



Ετερογενής προγραμματισμός

Το προγραμματιστικό μοντέλο του CUDA υποθέτει ότι τα threads εκτελούνται σε μια φυσικά διαχωριζόμενη συσκευή(device) η οποία λειτουργεί σαν συνεπεξεργαστής(εδώ η GPU) του βασικού σειριακού προγράμματος που εκτελείται στην CPU(host). Μόνο τα κομμάτια του προγράμματος που υλοποιούνται παράλληλα εκτελούνται στην GPU.



Υπολογιστική ικανότητα (Compute Capability)

Η υπολογιστική ικανότητα μιας συσκευής αντιπροσωπεύεται από έναν ειδικό αριθμό έκδοσης(ID). Αυτός ο αριθμός έκδοσης προσδιορίζει τις λειτουργίες που υποστηρίζονται από το υλικό GPU και χρησιμοποιείται από εφαρμογές κατά τη διάρκεια εκτέλεσης για να προσδιορίσει ποιες λειτουργίες υλικού είναι διαθέσιμες στην παρούσα GPU. Συσκευές με τον ίδιο αριθμό υπολογιστικής ικανότητας είναι της ίδιας αρχιτεκτονικής.



Οδηγίες υψηλών επιδόσεων

Για την βελτιστοποίηση της απόδοσης ακολουθούνται τρεις βασικές στρατηγικές:

- Μεγιστοποίηση της παράλληλης εκτέλεσης με σκοπό την μέγιστη χρησιμοποίηση της GPU
- Βελτιστοποίηση της χρησιμοποιούμενης και μεταφερόμενης μνήμης για την επίτευξη μέγιστου ρυθμού απόδοσης μνήμης
- Βελτιστοποίηση χρήσης εντολών για την επίτευξη μέγιστου ρυθμού απόδοσης εντολών

Οι στρατηγικές που θα αποφέρουν το καλύτερο κέρδος απόδοσης για ένα συγκεκριμένο τμήμα μιας εφαρμογής εξαρτώνται από τους περιορισμούς απόδοσης. Η βελτιστοποίηση της χρήσης εντολών ενός kernel περιορίζεται κυρίως από τις προσβάσεις στη μνήμη. Κατά συνέπεια, οι προσπάθειες βελτιστοποίησης θα πρέπει να κατευθύνονται συνεχώς με τη μέτρηση και την παρακολούθηση της απόδοσης, χρησιμοποιώντας για παράδειγμα τον Nvidia CUDA Profiler.

Για να μεγιστοποιηθεί η αξιοποίηση, η εφαρμογή θα πρέπει να είναι δομημένη με τέτοιο τρόπο ώστε να εκθέτει όσο το δυνατόν περισσότερο παραλληλισμό και να τον αντιστοιχίζει αποτελεσματικά στα διάφορα συστατικά του συστήματος, κρατώντας τα απασχολημένα για όσο περισσότερο χρόνο γίνεται. Σε κάθε επεξεργαστή πρέπει να ανατίθεται η εργασία στην οποία ανταποκρίνεται καλύτερα. Για την GPU(device) της εργασίες που εκτελούνται παράλληλα, ενώ για την CPU(host) τα σειριακά κομμάτια. Σε σημεία στον παράλληλο αλγόριθμο όπου ο παραλληλισμός έχει "σπάσει" επειδή ορισμένα νήματα πρέπει να συγχρονιστούν για να μοιραστούν τα δεδομένα μεταξύ τους, υπάρχουν δύο περιπτώσεις: Αν ανήκουν στο ίδιο μπλοκ χρησιμοποιείται η εντολή `__syncthreads()` και μοιράζονται δεδομένα μέσω της γρήγορης κοινής μνήμης μέσα στην ίδια κλήση του πυρήνα.

Αλλιώς πρέπει να μοιράζονται δεδομένα μέσω της καθολικής μνήμης χρησιμοποιώντας δύο ξεχωριστές κλήσεις πυρήνα, μία για εγγραφή και μία για ανάγνωση από την καθολική μνήμη. Η δεύτερη περίπτωση είναι λιγότερο βέλτιστη, δεδομένου ότι προσθέτει την επιβάρυνση των πρόσθετων κλήσεων του πυρήνα και της καθολικής κυκλοφορίας μνήμης. Επομένως, η εμφάνισή του θα πρέπει να ελαχιστοποιείται αντιστοιχίζοντας το πρόβλημα στο μοντέλο προγραμματισμού CUDA. Αυτό θα γίνει με τέτοιο τρόπο ώστε οι υπολογισμοί που απαιτούν επικοινωνία μεταξύ νημάτων να εκτελούνται, όπου είναι δυνατόν, μέσα σε ένα μπλοκ από νήματα.



Πολλαπλασιασμός Πινάκων σε GPU

Ένα ενδιαφέρον πρόβλημα που μπορεί να μελετηθεί με υπόβαθρο γνώσεων τις γλώσσας C για εξοικείωση με τον παράλληλο προγραμματισμό είναι ο πολλαπλασιασμός δύο τετράγωνων μητρώων(matrices) διάστασης n επί n .Προτείνετε η αυξητική εξέλιξη(incremental growth) για κάθε πρότζεκτ. Για τον λόγο αυτόν στην αρχή μπορείτε να δοκιμάσετε πίνακες 3X3.Για παράδειγμα αν A και B οι πίνακες που θέλουμε να πολλαπλασιάσουμε [3]:

$$\mathbf{A} = \begin{pmatrix} a & b & c \\ p & q & r \\ u & v & w \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \alpha & \beta & \gamma \\ \lambda & \mu & \nu \\ \rho & \sigma & \tau \end{pmatrix},$$

Τότε το γινόμενο τους AB θα είναι :

$$\mathbf{AB} = \begin{pmatrix} a & b & c \\ p & q & r \\ u & v & w \end{pmatrix} \begin{pmatrix} \alpha & \beta & \gamma \\ \lambda & \mu & \nu \\ \rho & \sigma & \tau \end{pmatrix} = \begin{pmatrix} aa + b\lambda + cp & a\beta + b\mu + c\sigma & a\gamma + b\nu + c\tau \\ p\alpha + q\lambda + r\rho & p\beta + q\mu + r\sigma & p\gamma + q\nu + r\tau \\ u\alpha + v\lambda + w\rho & u\beta + v\mu + w\sigma & u\gamma + v\nu + w\tau \end{pmatrix},$$

Ο βασικός κορμός του προγράμματος πολλαπλασιασμού θα είναι :

- Ορίζω τους πίνακες A και B και έναν πίνακα C που θα εκχωρηθεί το αποτέλεσμα και δεσμεύω χώρο στην κύρια μνήμη(memory allocation)
- Αρχικοποιώ τους πίνακες A και B με τυχαίες τιμές
- Ορίζω τους πίνακες d_A , d_B , d_C και δεσμεύω την αντίστοιχη global μνήμη της GPU
- Αντιγράφω τα δεδομένα από τους πίνακες A και B στην device memory με την χρήση της συνάρτησης cudaMemcpy()
- Ορίζω των αριθμό των μπλοκ και των αριθμό των νημάτων ανά μπλόκ με τα οποία θα καλέσω την συνάρτηση πυρήνα(kernel) , που στην περίπτωση μας είναι η συνάρτηση πολλαπλασιασμού μητρώων.
- Καλώ την συνάρτηση πυρήνα , η οποία θα εκτελεστεί στους πυρήνες της κάρτας γραφικών στην οποία θα γίνουν οι υπολογισμοί. Kernel<<<#blocks,#threadsPerBlock>>>(parameters)
- Αφού ολοκληρωθούν οι υπολογισμοί ο πίνακας d_C θα έχει τα επιθυμητά αποτελέσματα τα οποία αντιγράφω στον πίνακα C της κύριας μνήμης με την χρήση της συνάρτησης cudaMemcpy()

Για να μπορέσουμε να αξιολογήσουμε τα αποτελέσματα υπολογίζονται στην κάρτα γραφικών υλοποιώ τον αλγόριθμο πολλαπλασιασμού πινάκων και σειριακά και συγκρίνω τα αποτελέσματα με μια συνάρτηση υπολογισμού σφάλματος. Στο παράδειγμα αυτό χρησιμοποιήθηκε η συνάρτηση του μέσου τετραγωνικού σφάλματος. Επίσης μετρήθηκαν οι χρόνοι σε CPU και GPU για σύγκριση και υπολογισμό πιθανής επιτάχυνσης (Speedup).

Για να αντιμετωπιστούν προβλήματα περάσματος παραμέτρων σε συναρτήσεις δεν χρησιμοποιήθηκε δομή δεδομένων πίνακα 2 διαστάσεων αλλά μια δομής struct :

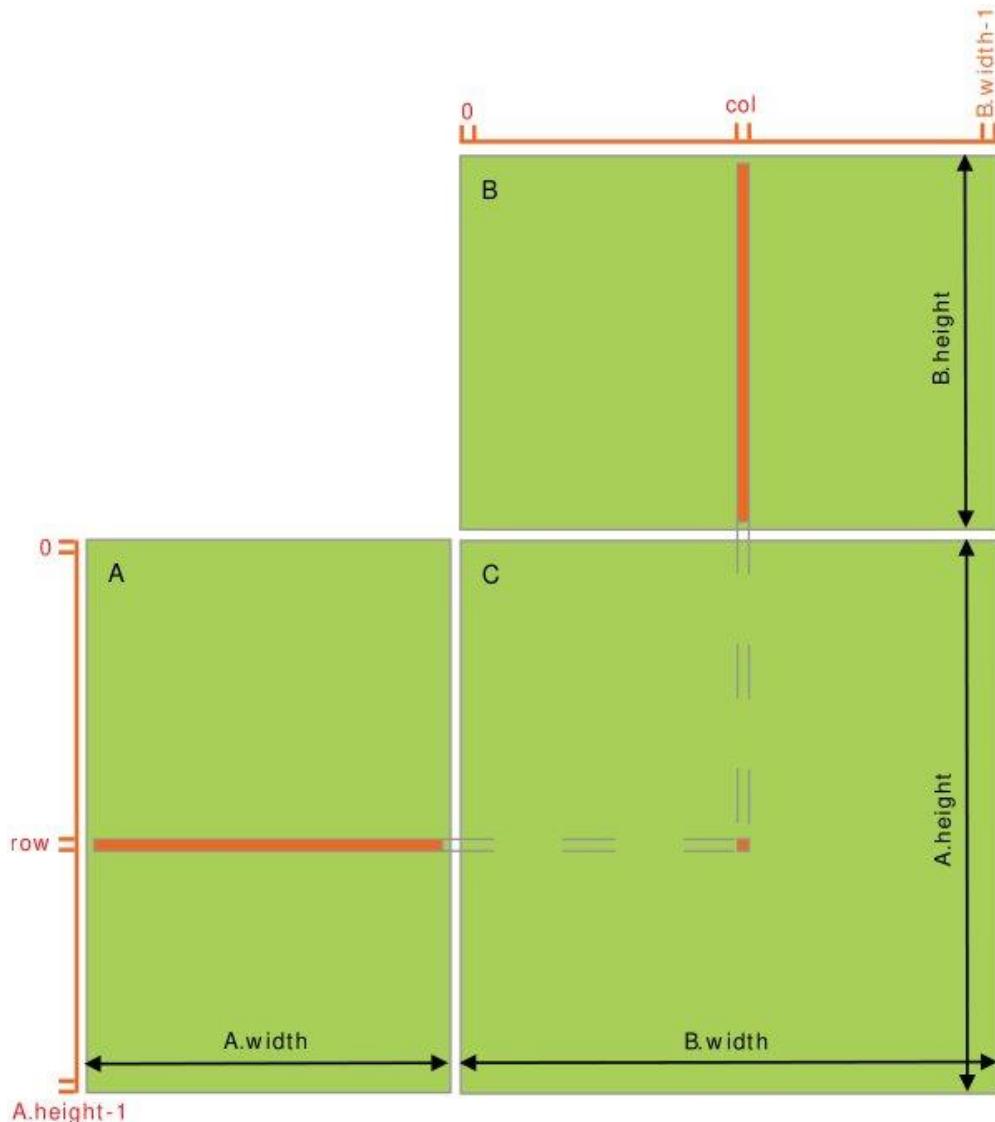
```
typedef struct {
    int width;
    int height;
    float* elements;
} Matrix;
```



Δεδομένου ότι τα δεδομένα αποθηκεύονται στην μνήμη με διάταξη σειράς (row-major order) στην γλώσσα C μπορούμε να διασχίσουμε σαν δυσδιάστατο πίνακα με τον παρακάτω μετασχηματισμό :
 $M(row, col) = *(M.elements + row * M.width + col)$.

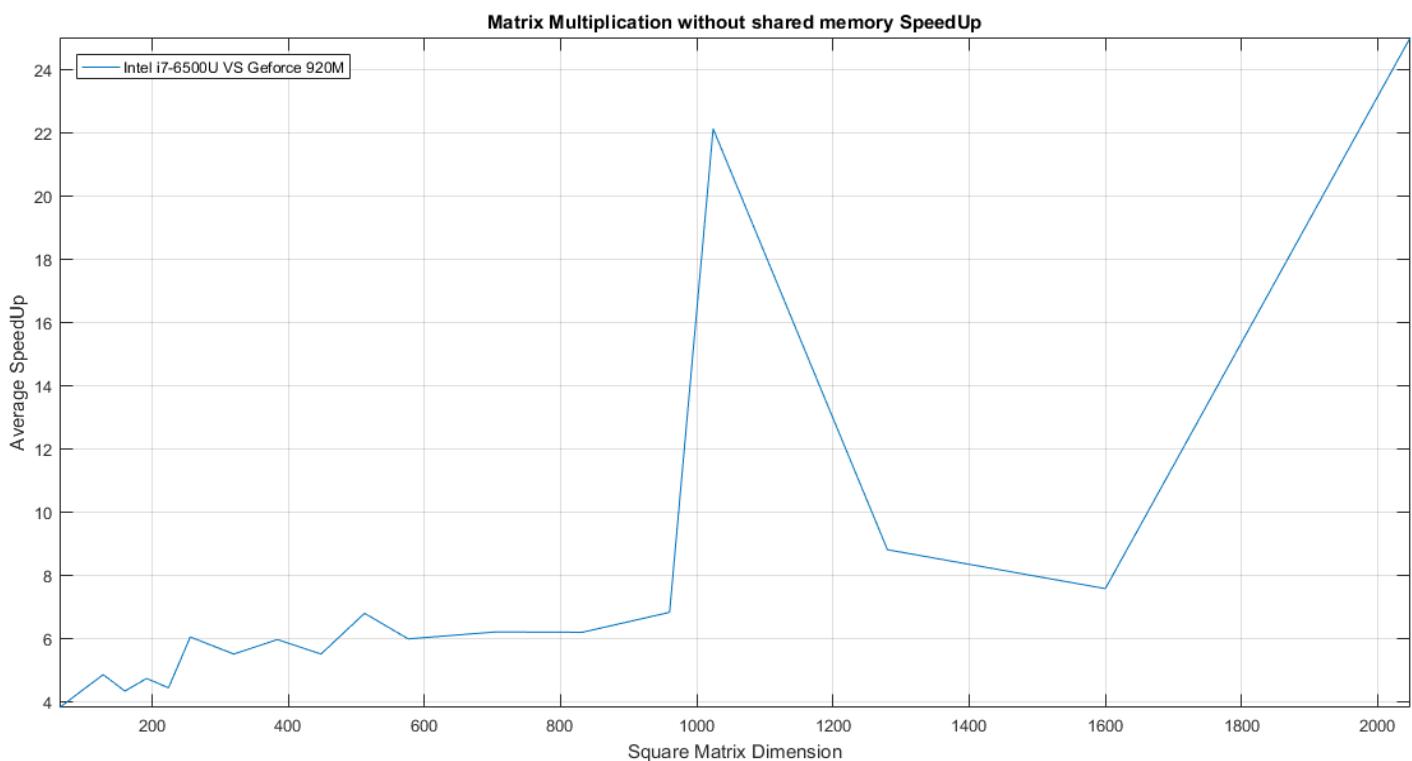
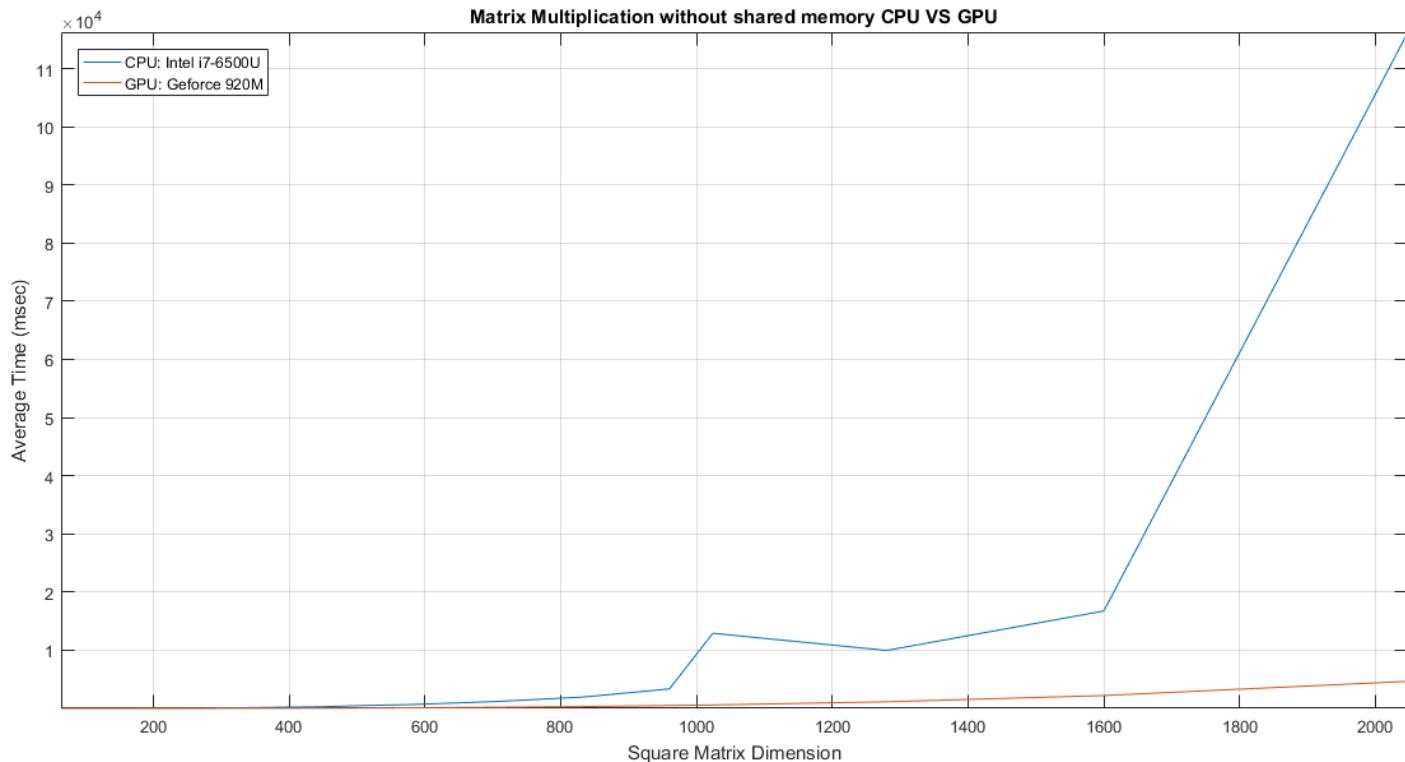
Οι κάρτες γραφικών μπορούν να υποστηρίξουν ένα συγκεκριμένο αριθμό threads per block , συνήθως 1024. Ενδείκνυται, σύμφωνα με την τεχνική διαιρεί και βασίλευε, τα νήματα να οργανωθούν σε όσο το δυνατόν περισσότερες ομάδες, δηλαδή blocks of threads. Για παράδειγμα ο πολλαπλασιασμός 2 τετραγωνικών πινάκων διάστασης 2048 δεν μπορεί να υλοποιηθεί χωρίς να χρησιμοποιηθούν block από νήματα. Για την καλύτερη αξιοποίηση από το hardware είναι προτιμότερο να αξιοποιούνται όσο το δυνατόν περισσότερα blocks ώστε η Gru να κατανέμει τον υπολογιστικό φόρτο βέλτιστα.

Μια πρώτη υλοποίηση του πολλαπλασιασμού πινάκων παρουσιάζεται στο διπλανό σχήμα. Κάθε νήμα υπολογίζει ένα στοιχείο του νέου πίνακα C . Διαβάζει μια γραμμή του πίνακα A από την global μνήμη και την αντίστοιχη στήλη του πίνακα B. Για τον υπολογισμό του στοιχείου του νέου πίνακα γίνεται πολλαπλασιασμός των δυο αυτών διανυσμάτων. Το αποτέλεσμα αποθηκεύεται στην global μνήμη του C. Στο συγκεκριμένο πρόβλημα δεν δημιουργούνται προβλήματα συγχρονισμού διότι τα νήματα διαβάζουν δεδομένα από κοινή μνήμη αλλά κάθε νήμα γράφει σε ένα εξωριστό στοιχείο του νέου μητρώου.



Παρακάτω παρουσιάζονται οι μετρήσεις χρόνου και επιτάχυνσης που έγιναν για την συγκεκριμένη υλοποίηση πολλαπλασιασμού πινάκων μεταξύ CPU και GPU.

Μέτρηση χρόνου και επιτάχυνσης

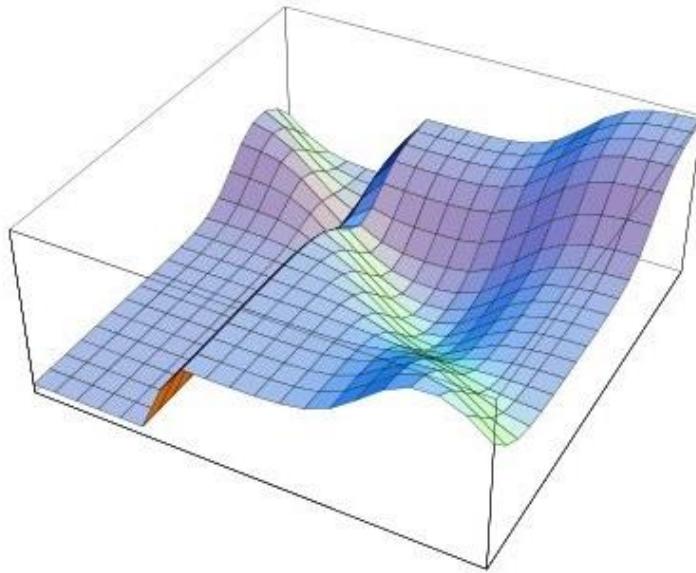


3. Άλλες μέθοδοι εκπαίδευσης

Προβλήματα της κλασσικής μεθόδου

Ένας από του λόγους που τα νευρωνικά δίκτυα απέκτησαν ξανά επιστημονικό ενδιαφέρον τα τελευταία 15 χρόνια , είναι η ανάπτυξη νέων μεθόδων που κατάφεραν να ανταποκριθούν καλύτερα σε προβλήματα μεγάλης κλίμακας. Από την ανακάλυψη του back propagation(BPA) το 1980 έχουν αναπτυχθεί πολλές εναλλακτικές μέθοδοι με διαφορετικά χαρακτηριστικά η κάθε μια . Ο βασικός στόχος όλων των νέων αλγορίθμων εκπαίδευσης είναι η καλύτερη και ταχύτερη σύγκλιση έναντι της κλασσικής μεθόδου.

Ο BPA παρουσιάζει , σε ορισμένα σύνολα δεδομένων (datasets) προβλήματα στην σύγκλιση . Αυτό συμβαίνει διότι τα N χαρακτηριστικά ενός συνόλου δεδομένων στον χώρο R^N να μην είναι διαχωρίσιμα . Επίσης η συνάρτηση σφάλματος ως προς του συντελεστές βαρύτητας μπορεί να έχει μεγάλο αριθμό ακροτάτων και ο αλγόριθμος να εγκλωβίζεται σε κάποια από αυτά με αποτέλεσμα να μην συγκλίνει σε επιθυμητά ακρότατα. Πολλές φορές έχει παρατηρηθεί ότι μπορεί να μην συγκλίνει και καθόλου όσο και να αυξήσουμε τις επαναλήψεις εκπαίδευσης. Ένα τρισδιάστατο παράδειγμα φαίνεται στην διπλανή εικόνα.



A local minimum of the error function

Για τους παραπάνω λόγους πολλοί ερευνητές των νευρωνικών δικτύων έχουν ασχοληθεί με την δημιουργία διάφορων παραλλαγών, με στόχο την ταχύτερη σύγκλιση. Παρακάτω θα γίνει μνεία των τεχνικών που μελετήθηκαν και υλοποιήθηκαν στην βιβλιοθήκη που αναπτύχθηκε στα χρησιμοποιούμενα σύνολα δεδομένων.

1. Οπισθοδρομική διάδοση με αδράνεια (momentum BPA)
2. Γρήγορη διάδοση σφάλματος (Quick-Prob)
3. Ελαστική οπισθοδρομική διάδοση σφάλματος(resilient BPA)

Για την δημιουργία των νέων μεθόδων οι μελετητές χρησιμοποίησαν μεθόδους μη-γραμμικής βελτιστοποίησης. Όλες οι νέες μέθοδοι που δοκιμάζονται συγκρίνονται με τον κλασσικό αλγόριθμο, που είναι αργός , διότι απαιτεί μεγάλη υπολογιστική πολυπλοκότητα . Γ' αυτό οι εναλλακτικές υλοποιήσεις καταγράφουν μια σημαντική βελτίωση. Από την άλλη πλευρά εφόσον έχει αποδειχθεί ότι το πρόβλημα της εκπαίδευσης για ένα δομημένο νευρωνικό δίκτυο είναι NP-complete , η υπολογιστική

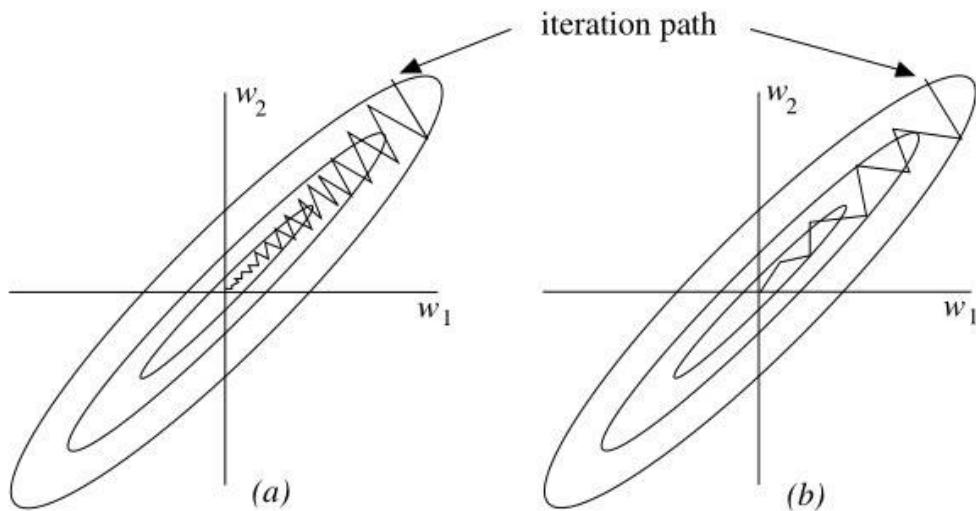


προσπάθεια στον υπολογισμό των παραμέτρων αυξάνεται εκθετικά με των αριθμό των αγνώστων. Παρόλα αυτά έχει παρατηρηθεί πειραματικά ότι ο standard BPA έχει καλύτερες επιδόσεις σε ορισμένα προβλήματα. Πιο συγκεκριμένα, όταν το πρόβλημα έχει μια ρεαλιστική πολυπλοκότητα και το μέγεθος των παραδειγμάτων εκπαίδευσης είναι πάνω από ένα κρίσιμο κατώφλι δίνει καλύτερα αποτελέσματα από τις περισσότερες νέες και πιο γρήγορες μεθόδους. Η ανάλυση που θα γίνει παρακάτω περιγράφει τα βασικά χαρακτηριστικά κάθε μεθόδου. Υπενθυμίζεται ότι μελετώνται μόνο δίκτυα που είναι οργανωμένα σε επίπεδα.

Αξίζει να αναφερθεί ότι στον επιστημονικό χώρο μελετώνται αλγόριθμοι εκπαίδευσης που θα είναι ικανοί να αλλάζουν την τοπολογία του δικτύου αποδοτικά. Κάτι τέτοιο, θα δώσει νέες δυνατότητες στα νευρωνικά δίκτυα προσαρμοζόμενων παραμέτρων και τοπολογίας και θα απαιτούνται πιο πολύπλοκοι αλγόριθμοι εκπαίδευσης.

Οπισθοδρομική διάδοση με αδράνεια (momentum propagation)

Η πρώτη βελτιωμένη παραλλαγή είναι ο αλγόριθμος με αδράνεια. Αδράνεια σημαίνει ότι δεν βασίζεται μόνο στην πληροφορία της αρνητικής μερικής παραγωγού αλλά εισάγεται ένας νέος που εκφράζει αδράνεια. Όταν το ελάχιστο της συνάρτησης σφάλματος βρίσκεται σε μια παραβολή (valley), ακολουθώντας την μέθοδο gradient descent μπορεί να οδηγηθεί σε μεγάλες ταλαντώσεις γύρω από την επιθυμητή τιμή. Η πρώτη και πιο απλή τροποποίηση είναι η εισαγωγή του συντελεστή αδράνειας. Στο σχήμα παρακάτω φαίνεται ένα διαισθητικό παράδειγμα ενός δικτύου με δύο βάρη.



Backpropagation without (a) or with (b) momentum term

Παρατηρώντας το μονοπάτι εξέλιξης του αλγορίθμου κατά την διάρκεια των επαναλήψεων στο δισδιάστατο επίπεδο των βαρών, συμπεραίνουμε ότι μπορούμε να μειώσουμε τις ταλαντώσεις. Η καλύτερη στρατηγική στην περίπτωση αυτή είναι να κατευθύνουμε την αναζήτηση προς το κέντρο της 'κοιλάδας' (valley), αλλά η μορφή της συνάρτησης σφάλματος δεν δείχνει προς την επιθυμητή κατεύθυνση. Εισάγοντας τον όρο της αδράνειας, η μεταβολή των συντελεστών βαρύτητας όχι μονό την τοπική κλίσης της παραγώγου αλλά και την προηγούμενη μεταβολή που υπολογίστηκε στην



προηγούμενη επανάληψη. Θεωρητικά, αυτή η προσέγγιση πρέπει να παρέχει στη διαδικασία αναζήτησης ένα είδος αδράνειας ικανό να βιοθήσει στην αποφυγή των υπερβολικών ταλαντώσεων στις στενές κοιλάδες της συνάρτησης σφάλματος .

$$\Delta w_{(t)} = -\epsilon \frac{\partial E}{\partial w_{(t)}} + \alpha \Delta w_{(t-1)}$$

Τονίζεται ότι ο όρος αδράνειας(α) και ο ρυθμός εκμάθησης(ϵ) συμβάλλουν στην επιτυχία της εκπαίδευσης, αλλά δεν αποτελούν μέρος του νευρικού δικτύου.

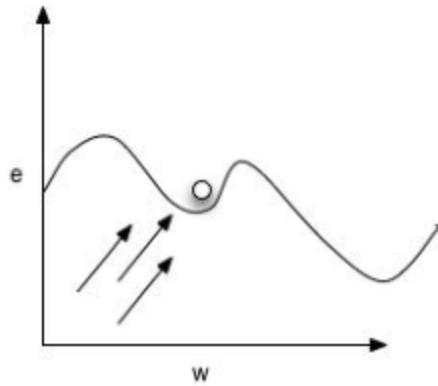
Μπορείτε να χρησιμοποιήσετε την αδράνεια για να καταπολεμήσετε τα τοπικά ελάχιστα. Αν βρείτε το νευρωνικό δίκτυο σε στασιμότητα, μια υψηλότερη τιμή αδράνειας θα μπορούσε να ωθήσει την εκπαίδευση πέρα από το τοπικό ελάχιστο που αντιμετώπισε. Τελικά, η επιλογή καλών τιμών για την αδράνεια και το ρυθμό εκμάθησης είναι μια διαδικασία διαδοχικών δοκιμών με προσαρμογές ανάλογα με το σφάλμα(trial and error). Δεδομένου ότι οι βέλτιστες παράμετροι εξαρτώνται σε μεγάλο βαθμό από το συγκεκριμένο πρόβλημα, δεν έχει αναπτυχθεί γενική στρατηγική για την αντιμετώπιση αυτού του προβλήματος των δοκιμών. Επομένως, παρουσιάζονται τα συμπεράσματα σχετικά με την επιλογή ενός συγκεκριμένου ρυθμού μάθησης και αδράνειας, και το είδος της ταλαντεύομενης συμπεριφοράς που μπορεί να παρατηρηθεί με τον κανόνα ανατροφοδότησης backpropagation και τους μεγάλους ρυθμούς αδράνειας.

Μπορείτε να μεταβάλλετε και του δυο συντελεστές. Ενδείκνυται να κρατάτε ο ένας από τους δύο σταθερός και να μεταβάλλετε ο άλλος για την εξαγωγή καλύτερων και πιο ασφαλών συμπερασμάτων. Η αδράνεια ρυθμίζεται συχνά από 0.5 μέχρι 0.9 και ο ρυθμός εκμάθησης είναι 0,1 ή χαμηλότερος. Πολλοί επιστήμονες , στην προσπάθεια να βρουν συσχετίσεις μεταξύ των δύο συντελεστών χρησιμοποιούν τον τύπο :

$$\epsilon = K(1 - \alpha)$$

Οπού το K είναι μια σταθερά από 0.5 έως 1 . Γενικά μικρότεροι αριθμοί κρυμμένων νευρώνων θα πρέπει να χρησιμοποιούν μεγαλύτερο K .Με αυτόν τον τρόπο απαιτείται ο προσδιορισμός μόνο της παραμέτρου α για την εκπαίδευση και καταλήγουμε με λιγότερες δοκιμές στο συμπέρασμα της αποδοτικής εκπαίδευσης. Ένα απλό παράδειγμα των προβλημάτων που ξεπερνούνται δίνεται στο σχήμα :





Ενώ η κλίση της παραγώγου τείνει στο 0 ο όρος της αδράνειας ωθεί την τιμή του συντελεστή σε μια τοπικά μεγαλύτερη τιμή σφάλματος με αποτέλεσμα να αποφεύγονται τα τοπικά ελάχιστα.

Γρήγορη μετάδοση σφάλματος(Qprob)

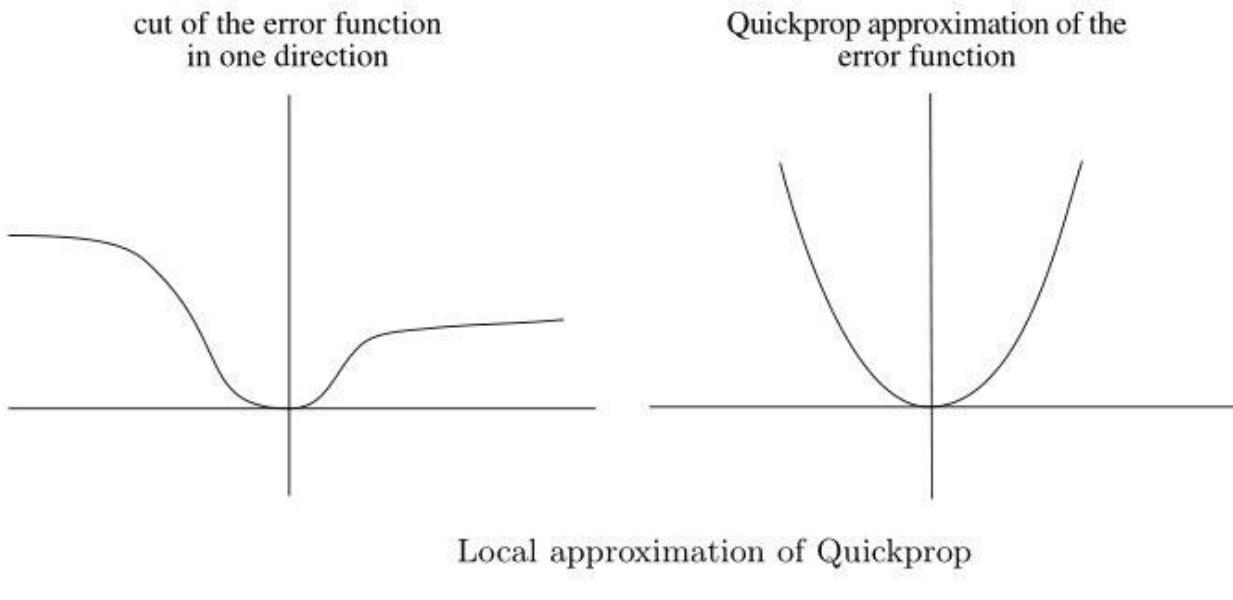
Ο αλγόριθμος γρήγορης διάδοσης (**Qprob**) ανήκει στην οικογένεια των αλγορίθμων δεύτερης τάξης. Η οικογένεια αλγορίθμων δεύτερης τάξης χρησιμοποιεί περισσότερες πληροφορίες σχετικά με την μορφή της συνάρτησης σφάλματος από την τοπική τιμή της κλίσης. Μια καλύτερη επανάληψη μπορεί να γίνει εάν η καμπυλότητα της συνάρτησης σφάλματος θεωρείται επίσης σε κάθε βήμα, πληροφορία που λαμβάνεται από την δεύτερη παράγωγο. Σε μεθόδους δεύτερης τάξης χρησιμοποιείται τετραγωνική προσέγγιση της συνάρτησης σφάλματος. Οι συντελεστές για την επανάληψη k θα άλλαζαν :

$$w_i^{(k+1)} = w_i^{(k)} - \frac{\nabla_i E(\mathbf{w})}{\partial^2 E(\mathbf{w}) / \partial w_i^2}.$$

Όμως η πληροφορίας δεύτερης τάξης απαιτεί πολλούς υπολογισμούς . Στα νευρωνικά δίκτυα πρέπει να επαναλάβουμε αυτόν τον υπολογισμό σε κάθε επανάληψη. Ανάλογα το πόσα δεδομένα έχουμε στην διάθεση μας, μπορεί να χρειαστούν από 2 μέχρι 300 χιλιάδες επαναλήψεις , πολλές φορές και περισσότερες. Επιπλέον, οι μέθοδοι δεύτερης τάξης λειτουργούν καλά όταν η λειτουργία σφάλματος έχει τετραγωνική μορφή, αλλιώς είναι πιθανό μικρή τιμή της δεύτερης παράγωγου, να οδηγήσει σε εξαιρετικά μεγάλες αλλαγές. Κατά συνέπεια, έχουν προταθεί πολλές τεχνικές για την προσέγγιση των πληροφοριών δεύτερης τάξης καθώς και της αντιμετώπιση μεγάλων αλλαγών. Μια από αυτές και ίσως η πιο απλή είναι ο αλγόριθμος Qprob.

Ο αλγόριθμος Qprob προσπαθεί να λάβει υπόψη πληροφορίες δεύτερης τάξης ακολουθώντας μία σχετικά απλή προσέγγιση: λαμβάνονται μόνο τοπικά βήματα ελαχιστοποίησης. Γι' αυτό συχνά αναφέρεται στην βιβλιογραφία ως 'μέθοδος Φευτό-Newton Raphson'. Οι πληροφορίες σχετικά με την καμπυλότητα της συνάρτησης σφάλματος υπολογίζονται από την τρέχον και την προηγούμενη τιμή της μερικής παραγώγου.





Το βασικό πλεονέκτημα του αλγορίθμου είναι ότι είναι ανεξάρτητος για κάθε βάρος w , δηλαδή είναι μια τοπική στρατηγική. Αντίθετα ο συντελεστής εκπαίδευσης που εφαρμόζεται στον κλασσικό αλγόριθμο, όπως και ο όρος της αδράνειας είναι γενικές στρατηγικές. Χρησιμοποιείται για τον όρο δεύτερης τάξης μια μονοδιάστατη προσέγγιση της συνάρτησης σφάλματος. Συνεπώς η αναπροσαρμογή γίνεται με τον εξής τύπο :

$$\Delta^{(k)} w_i = \Delta^{(k-1)} w_i \left(\frac{\nabla_i E^{(k)}}{\nabla_i E^{(k-1)} - \nabla_i E^{(k)}} \right)$$

Παρατηρείται ότι οι μόνες επιπλέον πληροφορίες που χρειάζεται να αποθηκεύονται στην μνήμη είναι η προηγούμενη αλλαγή του W_i , καθώς και η προηγούμενη τοπική κλίση. Η χωρική πολυπλοκότητα δεν αυξάνεται πολύ, ενώ χρονικά το κόστος κάθε επανάληψης δεν δημιουργεί σημαντική επιβάρυνση.

Το βασικό μειονέκτημα του αλγόριθμου είναι ότι μικρές τιμές των παραγωγών καθιστούν υπερβολικά μεγάλες αλλαγές στους συντελεστές βαρύτητας. Αυτό μπορεί να αποφευχθεί περιορίζοντας την τιμή Δw_i σε ένα κάτω κατώφλι. Με τον ίδιο τρόπο περιορίζουμε και για την διαφορά των παραγώγων στον παρανομαστή, ιδιαίτερα σε υλοποιήσεις με δεδομένα τύπου float, ώστε να μην πάρει πολύ μικρή τιμή.

Ελαστική διάδοση(Rprop)

Ένας νέος αλγόριθμος εκπαίδευσης- μάθησης για τα πολυεπίπεδα νευρωνικά δίκτυα είναι αυτός της ελαστικής διάδοσης(resilient propagation) . Για να αντιμετωπίσει τα εγγενή μειονεκτήματα της gradient descent ο Rprop εκτελεί μια τοπική προσαρμογή στις αλλαγές των συντελεστών βαρύτητας σύμφωνα με την συμπεριφορά της συνάρτησης σφάλματος. Η ουσιώδης του διαφορά είναι , σε σχέση με όλες τις υπόλοιπες μεθόδους μάθησης , δεν επηρεάζεται από το μέγεθος της ολικής παραγώγου , ενώ εξαρτάται από το πρόσημο της . Αυτή η μέθοδος χρησιμοποιήθηκε γιατί η παράγωγος μπορεί να



παίρνει απρόβλεπτες τιμές με αποτέλεσμα να καθυστερούν την σύγκλιση. Έτσι οδηγείται σε μια πιο ομαλή και αποδοτική διαδικασία προσαρμογής των συντελεστών.

Πολλές προσπάθειες έχουν καταβληθεί μέχρι τώρα για την προσαρμογή των παραμέτρων του νευρωνικού δικτύου κατά την διαδικασία της εκπαίδευσης. Έτσι οι αλγόριθμοι μπορούν να κατηγοριοποιηθούν σε γενικές και τοπικές στρατηγικές. Οι προσαρμογές με γενικές τεχνικές χρησιμοποιούν την γνώση που είναι αποθηκευμένη στην τρέχουσα κατάσταση όλου του δικτύου για την τροποποίηση του δικτύου. Αντίθετα, οι τοπικές στρατηγικές χρησιμοποιούν πληροφορία συσχετιζόμενη με το τοπικό βάρος, όπως η μερική παράγωγος.

Θα πρέπει επίσης να ληφθεί υπόψιν, ότι οι τοπικές στρατηγικές προσαρμογής είναι πιο στενά συνδεδεμένη με την ιδέα της μάθησης των πραγματικών βιολογικών νευρώνων και είναι πιο κατάλληλες για παράλληλες υλοποιήσεις. Είναι φανερή, υπεροχή των τοπικών στρατηγικών σε σχέση με τις γενικές τεχνικές (global).

Η πλειοψηφία και των δύο αλγορίθμων Εκτελεί μία αναπροσαρμογή Του ρυθμό μάθησης σύμφωνα με αυτά που παρατηρούνται η συνάρτηση σφάλματος. Αυτό που Συνήθως δεν λαμβάνεται υπόψη είναι ότι το βήμα κάθε βάρους δεν εξαρτάται μόνο το ρυθμό μάθησης αλλά Επίσης από τη μερική παραγωγό Του συντελεστή αυτού. Έτσι προσπαθώντας να αναπροσαρμόζουμε προσεχτικά (με όσο το δυνατόν καλύτερο τρόπο) το συντελεστή βαρύτητας Ενώ η μερική παραγωγός έχει τελείως απρόβλεπτη συμπεριφορά Δεν οδηγούμαστε σε επιθυμητά αποτελέσματα. Αυτός ήταν ένας από τους βασικούς λόγους της ανάπτυξης του αλγόριθμου Rprop Χωρίς να λαμβάνεται υπόψη το μέτρο της μερικής παραγωγού του κάθε νευρώνα. Επίσης εξασφάλισε Τη μεταβολή των συντελεστών δεν θα παίρνει ακραίες τιμές.

Έχουν προταθεί πολλές παραλλαγές του αλγόριθμου Οπισθοδρομική διάδοσης του σφάλματος Με στόχο να επιταχύνει σε σημαντικό βαθμό την ταχύτητα της σύγκλισης. Οι περισσότεροι από αυτούς δούλευαν ελάχιστα καλύτερα σε πολύ λίγα προβλήματα ενώ στα υπόλοιπα δεν βελτίωναν τη σύγκριση ή ακόμα έδιναν χειρότερα αποτελέσματα.

Για να αποφύγει λοιπόν την χρήση του μέτρου της μερικής παραγωγού για κάθε συντελεστή εισάγει ένα νέο όρο Δij που καθορίζει τη μεταβολή του βάρους Δwij. Αυτός ο όρος ανανεώνεται στη διαδικασία της εκπαίδευσης σύμφωνα με την τοπική τιμή της συνάρτησης σφάλματος με τον παρακάτω τύπο :

$$c = \frac{\partial E}{\partial w_{ij}}^{(t)} \cdot \frac{\partial E}{\partial w_{ij}}^{(t-1)}$$

$$\Delta_{ij}^{(t)} = \begin{cases} \eta^+ \cdot \Delta_{ij}^{(t-1)}, & \text{if } c > 0 \\ \eta^- \cdot \Delta_{ij}^{(t-1)}, & \text{if } c < 0 \\ \Delta_{ij}^{(t-1)}, & \text{otherwise} \end{cases}$$

Ο κανόνας αυτός δουλεύει ως εξής: Κάθε φορά που η αντίστοιχη μερική παραγωγός αλλάζει πρόσωπο, το οποίο υποδεικνύει ότι η τελευταία ενημέρωση ήταν πολύ μεγάλη και ο αλγόριθμος πιθανότητα πέρασε από κάποιο τοπικό ελάχιστο. Τότε όρος Δij μειώνεται κατά τον παράγοντα η-. Αντίθετα αν η παραγωγός διατηρεί το πρόσωπό της τότε όρος Δij αυξάνεται ελάχιστα κατά τον παράγοντα η+, με σκοπό με την ταχύτερη σύγκλιση σε αβαθείς περιοχές. Για την υλοποίηση αυτήν απαιτείται η αποθήκευση για κάθε συντελεστή βαρύτητας wij η αποθήκευση της προηγούμενης κλίσης (μερικής παραγώγου).



Εν συνεχεία η μεταβολή του συντελεστή βαρύτητας ακολουθεί τον εξής κανόνα : Αν η παραγωγός είναι θετική δηλαδή το σφάλμα αυξάνεται τότε η μεταβολή του βάρους αφαιρείται κατά τον συντελεστή Δ_{ij} , ενώ αν είναι θετική προστίθεται .

$$\Delta w_{ij}^{(t)} = \begin{cases} -\Delta_{ij}^{(t)} & , \text{ if } c > 0 \\ +\Delta_{ij}^{(t)} & , \text{ if } c < 0 \\ 0 & , \text{ otherwise} \end{cases}$$

Η μοναδική εξαίρεση είναι όταν η μερική παραγωγός αλλάζει πρόσημο, δηλαδή το προηγούμενο βήμα του αλγόριθμου ήταν πολύ μεγάλο και το ελάχιστο προσπεράστηκε. Στην περίπτωση αυτή η αλλαγή του βάρους αναστρέφεται Σύμφωνα με την προηγούμενη τιμή της.

$$\Delta w_{ij}^{(t)} = -\Delta w_{ij}^{(t-1)}, \text{ if } \frac{\partial E}{\partial w_{ij}}^{(t-1)} * \frac{\partial E}{\partial w_{ij}}^{(t)} < 0$$

Για να μην επηρεάσει την επόμενη επανάληψη η αλλαγή που κάνουμε θέτουμε $\frac{\partial E}{\partial w_{ij}}^{(t-1)} := 0$

Ο αλγόριθμος όπως παρουσιάστηκε πρώτη φορά το 1994 :

For all weights and biases{

```

if ( $\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) > 0$ ) then {
     $\Delta_{ij}(t) = \text{minimum}(\Delta_{ij}(t-1) * \eta^+, \Delta_{max})$ 
     $\Delta w_{ij}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$ 
     $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$ 
}
else if ( $\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) < 0$ ) then {
     $\Delta_{ij}(t) = \text{maximum}(\Delta_{ij}(t-1) * \eta^-, \Delta_{min})$ 
     $w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1)$ 
     $\frac{\partial E}{\partial w_{ij}}(t) = 0$ 
}
else if ( $\frac{\partial E}{\partial w_{ij}}(t-1) * \frac{\partial E}{\partial w_{ij}}(t) = 0$ ) then {
     $\Delta w_{ij}(t) = -\text{sign}(\frac{\partial E}{\partial w_{ij}}(t)) * \Delta_{ij}(t)$ 
     $w_{ij}(t+1) = w_{ij}(t) + \Delta w_{ij}(t)$ 
}
}
```

Οι παράμετροι του αλγορίθμου:

Κατά την εκκίνηση οι αρχικές τιμές των Δ_{ij} είναι $\Delta 0$. Η επιλογή της τελευταίας παραμέτρου ενώ επηρεάζει τις αρχικές αλλαγές των συντελεστών βαρύτητας Έχει αποδειχθεί πειραματικά ότι δεν αλλάζει το τελική σύγκλιση για οποιαδήποτε αρχική επιλογή της παραμέτρου Αυτής Ούτε επηρεάζει την ταχύτητα σύγκλισης. Μία τυπική τιμή που χρησιμοποιείται ευρέως είναι το 0.1 . Οι παράμετροι Δ_{max} , Δ_{min} λαμβάνουν τιμές 50 και 0.0001 αντίστοιχα ελαχιστοποιώντας την πιθανότητα υπερχείλισης/υποχείλισης για υλοποίηση με μεταβλητές τύπου float. Τονίζεται ότι πολλές φορές η



παράμετρος Δmax αρχικοποιείται με μικρότερη τιμή (συνήθως 1), δίνοντας μια περισσότερο ομαλή συμπεριφορά στην μείωση του σφάλματος. Δεν είναι επιθυμητό οι συντελεστές να παίρνουν μεγάλες τιμές και με την αρχικοποίηση αυτή μπορεί να αποφευχθεί. Για υλοποιήσεις σε κάρτα γραφικών, βάσει προσωπικών δοκιμών ενδείκνυται η τιμή 1.

Όσον αφορά την επιλογή των παραμέτρων $\eta+$, $\eta-$, Αν η παραγωγός αλλάζει πρόσημο συμπεραίνουμε ότι η προηγούμενη αλλαγή ήταν πολύ μεγάλη. Επειδή δεν χρησιμοποιούμε την πληροφορία του μέτρου της παραγωγού δεν γνωρίζουμε κατά πόσον έχει αστοχήσει από το σφάλμα, χρησιμοποιούμε εμπειρικά την τιμή $\eta=0.5$. Η παράμετρος $\eta+$ οφείλει να είναι αρκετά μεγάλη ώστε να επιταχύνει τη σύγκλιση ειδικά σε αβαθείς περιοχές Και όχι ιδιαίτερα μεγάλη ώστε να αλλάζει συνέχεια το πρόσημο της μεταβολής των βαρών. Βάσει δοκιμών η επιλογή της τιμής $\eta+=1.2$ έδωσε τα καλύτερα πειραματικά αποτελέσματα ανεξαρτήτως του προβλήματος το οποίο καλούνταν να λύσει.

Ένα από τα κύρια πλεονεκτήματα του RPROP έγκειται στο γεγονός, ότι για πολλά προβλήματα δεν απαιτείται επιλογή των αρχικών παραμέτρων για να ληφθεί ο βέλτιστος ή σχεδόν βέλτιστος χρόνος σύγκλισης.

Ο βασικός λόγος επιτυχίας του αλγόριθμου (εκτός του γεγονότος ότι λαμβάνει μόνο υπόψη το πρόσημο της παραγωγού) είναι στον κλασικό αλγόριθμο, Το μέγεθος της παραγωγού μειώνεται εκθετικά από την απόσταση του νευρώνα σε σχέση με το επίπεδο εξόδου. Αυτό συμβαίνει λόγω της κλίσης των σιγμοειδών συναρτήσεων, Με αποτέλεσμα Όσο πιο μακριά είναι ένας συντελεστής από το επίπεδο εξόδου να επηρεάζεται όλο και λιγότερο δηλαδή να μαθαίνει πιο αργά. Χρησιμοποιώντας τον RPROP, επειδή λαμβάνονται υπόψη μόνο τα πρόσημα η μάθηση είναι ισοπίθανα κατανεμημένη σε όλο το νευρωνικό δίκτυο.

Σημαντικό πλεονέκτημα είναι ο αριθμός των επαναλήψεων που μπορεί να είναι από 5 μέχρι 25 φορές λιγότερες σε σχέση με την κλασική μέθοδο, ενώ το πρόσθετο κόστος υπολογισμού είναι σχετικά μικρό. Παρόλα αυτά πρέπει να ληφθούν υπόψιν και οι επιπλέον απαιτήσεις από άποψη μνήμης ειδικά για υλοποίησης σε GPU's. Ένα άλλο μείζον χαρακτηριστικό, με ιδιαίτερη σημασία σε πρακτικές εφαρμογές, είναι η δυναμικότητα του νέου αλγορίθμου κατά την επιλογή των αρχικών παραμέτρων του.



4. Υλοποίηση ΤΝΔ σε GPU

Αρχική υλοποίηση ΤΝΔ

Η πρώτη υλοποίηση που σχεδιάστηκε ήταν ένα στατικό ΤΝΔ με 3 επίπεδα και συγκεκριμένο αριθμό νευρώνων. Αυτό διευκολύνει την υλοποίηση, διότι δεσμεύουμε στατική, εφόσον κατά την μεταγλώττιση γνωρίζουμε τις παραμέτρους του ΤΝΔ. Λόγω της αβεβαιότητας που υπάρχει κατά τον προγραμματισμό σε GPU, ελέγχθηκε κάθε βήμα του αλγορίθμου κατά την διάρκεια εκπαίδευσης για ένα μόνο παράδειγμα. Ο έλεγχος έγινε αρχικοποιώντας του συντελεστές βαρύτητας με μια μικρή τιμή (0.1) και βλέποντας της έξοδο του κάθε επιπέδου. Για την εμφάνιση των αποτελεσμάτων τα ενδιάμεσα αποτελέσματα αντιγράφονται στην CPU για να εκτυπωθούν.

Όμοια για τον BPA, αντιγράφονταν τα δέλτα και η μεταβολές στους συντελεστές βαρύτητας και υπολογίστηκαν για επαλήθευση με τον αλγόριθμο σε CPU και μετρήθηκε το σφάλμα. Πολλές φορές χρειάζεται και ο χειρωνακτικός υπολογισμών για να βεβαιωθούμε ότι ο αλγόριθμος εκτελεί τις λειτουργίες που τον προγραμματίζουμε. Τέλος, ένα επιπλέον τρόπος να βεβαιωθούμε ότι λειτουργεί σωστά είναι να χρησιμοποιούμε πολλές φορές το ίδιο παράδειγμα για εκπαίδευση και να βλέπουμε το σφάλμα με την επιθυμητή τιμή να μειώνεται.

Γενίκευση ΤΝΔ με δυνατότητα ευελιξίας παραμέτρων

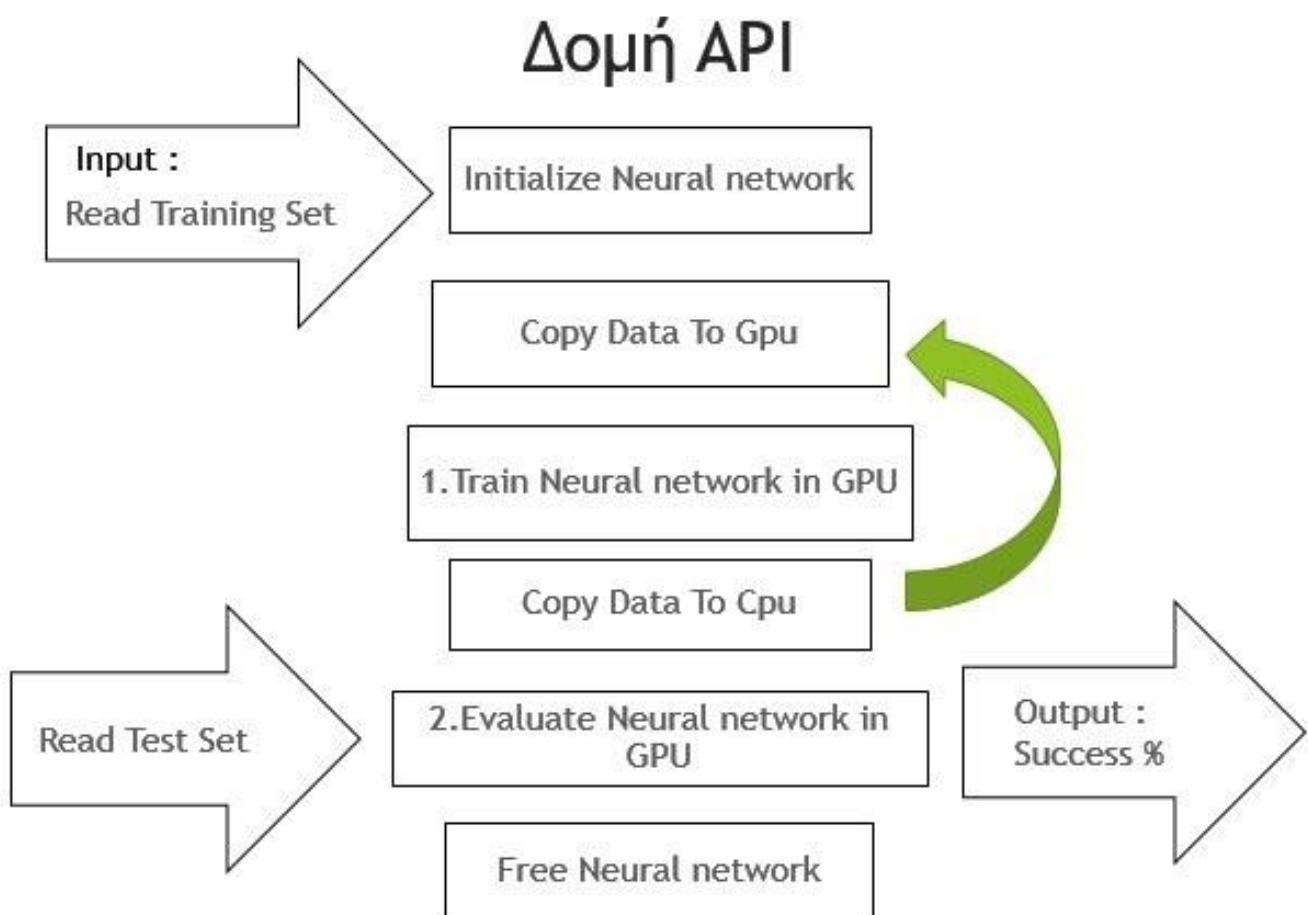
Εν συνεχείᾳ, εφόσον κατανοήθηκε καλύτερα η λειτουργία των αλγορίθμων, υλοποιήθηκε σε μορφή βιβλιοθήκης. Στόχος ήταν να λαμβάνει τις βασικές παραμέτρους και να δεσμεύει δυναμικά όλη την απαίτουμενη μνήμη δυναμικά(run time). Οι βασικές παράμετροι που καθορίζουν πλήρως την τοπολογία του δικτύου είναι: το μήκος της εισόδου, ο αριθμός των επιπέδων, και οι νευρώνες ανά επίπεδο. Τα δεδομένα αυτά είναι αρκετά για την δέσμευση της μνήμης και την αρχικοποίηση των βαρών. Για την εκπαίδευση χρειάζεται ο καθορισμός και άλλων παραμέτρων όπως ο ρυθμός εκπαίδευσης, ο αριθμός των επαναλήψεων, το ελάχιστο σφάλμα σύγκλισης.

Λόγω της πολυπλοκότητας που εισάγει η παραμετροποίηση της τοπολογίας από την χρήστη της βιβλιοθήκης, κρίθηκε απαραίτητο η ανάπτυξη μια δομής δεδομένων(struct) που να χαρακτηρίζει το ΤΝΔ. Η δομή αυτή περιέχει όλους του δείκτες σε CPU και GPU για την δημιουργία και την εκπαίδευση ενός ΤΝΔ. Σε προσέγγιση αντικειμενοστραφούς προγραμματισμού η δομή struct είναι μια κλάση. Ωστόσο, αξίζει να διευκρινιστεί ότι ένα ΤΝΔ μόνο καθορίζεται από την τοπολογία και τις τιμές των βαρών. Οι τιμές των εξόδων για κάθε επίπεδο, τα δέλτα κάθε νευρώνα είναι πληροφορία που χρησιμοποιείται κατά την εκπαίδευση και είναι διαφορετικά για κάθε επανάληψη. Το βήμα της γενίκευσης ήταν απατητικό προγραμματιστικά. Για τον έλεγχο χρησιμοποιήθηκε η προηγούμενη υλοποίηση με αρχικοποίηση όλων των βαρών με τιμή(0.1), για εκπαίδευση με ένα παράδειγμα.

Εφόσον έγινε ο έλεγχος σωστής λειτουργίας έγινε προσπάθεια ελαχιστοποίησης των μεταφορών μνήμης. Το τελευταίο επιτεύχθηκε αντιγράφοντας τα βάρη μια φορά κατά την εκκίνηση στην GPU και μια φορά κατά το τέλος της εκπαίδευσης. Επίσης, τα ενδιάμεσα αποτελέσματα όπως τα δέλτα και οι έξοδοι των νευρώνων δεν αποθηκεύονται στην CPU. Τέλος το σφάλμα της επιθυμητής τιμής εν συγκρίσει με την έξοδο ελέγχεται στην CPU ανά τακτά χρονικά διαστήματα (πχ ανά 100 επαναλήψεις) και μόνο τότε γίνεται η μεταφορά των δεδομένων.

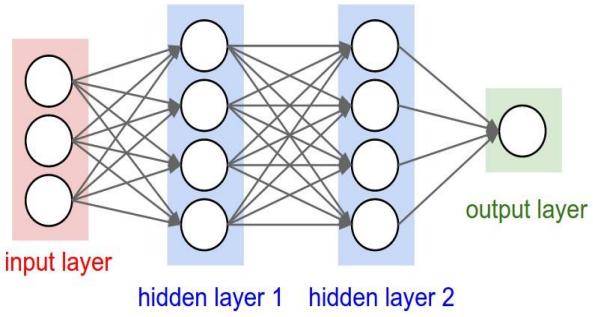


//description++++



Μελέτη παραλληλισμού εμπρόσθιας διάδοσης νευρωνικών δικτύων

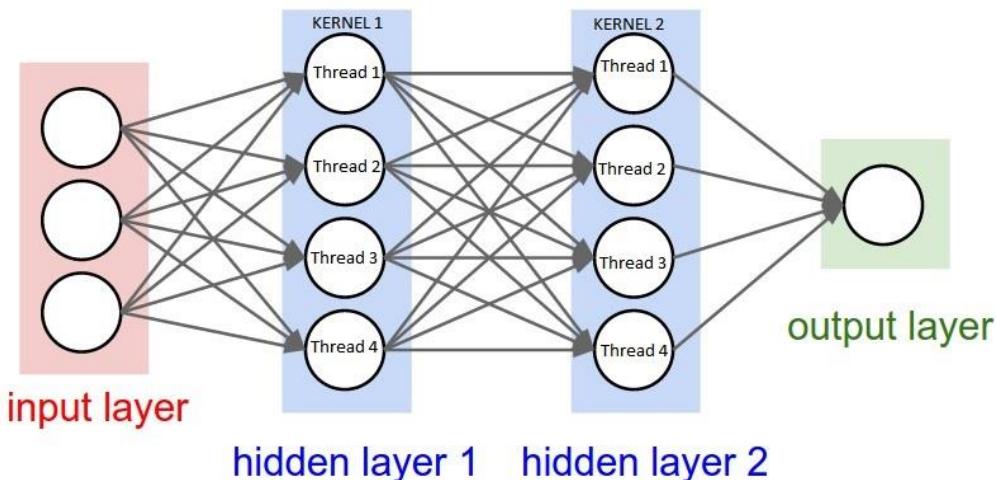
Τα TNΔ οργανώνονται σε επίπεδα και η είσοδος του κάθε επίπεδου είναι η έξοδος του προηγουμένου. Το πρώτο επίπεδο έχει ως είσοδο το διάνυσμα εισόδου του παραδείγματος εκπαίδευσης. Παρατηρείται λοιπόν, ότι υπάρχει εξάρτηση του ενός επίπεδου από τα προηγούμενα, ανεξαρτήτως υλοποίησης. Γ' αυτό τα επίπεδα δεν μπορούν, παρά να εκτελούνται σειριακά.



Το τελευταίο συμπέρασμα, αν και μη επιθυμητό, θα δούμε ότι δεν αποτελεί μεγάλο πρόβλημα, διότι δεν συνηθίζεται να χρησιμοποιούνται πολλά επίπεδα. Πιο συγκεκριμένα, για TNΔ χωρίς αραιές συνδέσεις τοποθετούνται μέχρι 4 επίπεδα στην αναγνώριση προτύπων. Αυτό βέβαια εξαρτάται και από τα δεδομένα που διαθέτουμε. Επομένως, στόχος είναι να χρησιμοποιήσουμε όσο λιγότερα επίπεδα είναι δυνατόν. Μια κλασσική στρατηγική είναι η εκκίνηση από 2 επίπεδα, 1 κρυφό και 1 έξοδου. Τα TNΔ με ένα επίπεδο μπορεί να αναγνωρίσουν μόνο γραμμικά παραδείγματα και δεν χρησιμοποιούνται.

Αντίθετα, κάθε τεχνητός νευρώνας σε ένα συγκεκριμένο επίπεδο, λειτουργεί τελείως ανεξάρτητα από του υπόλοιπους. Διαβάζουν μεν τα ίδια δεδομένα αλλά επεξεργάζονται τα δεδομένα και γράφουν σε διαφορετικές εξόδους. Το μοντέλο του νευρώνα επομένως, ανταποκρίνεται ιδανικά στο μοντέλο των καρτών γραφικών.

Η πρώτη υλοποίηση για την εμπρόσθια διάδοση έγινε θεωρώντας κάθε νευρώνα ως ένα thread της GPU και κάθε GPU Kernel ως ένα επίπεδο. [Παράρτημα A]



Το βασικό πλεονέκτημα στην παράλληλη υλοποίηση TNΔ είναι η τοποθέτηση όσων νευρώνων είναι επιθυμητό από τον χρήστη, χωρίς σχεδόν καθόλου χρονική καθυστέρηση. Ο μοναδικός περιορισμός είναι ο αριθμός των threads που μπορεί να εκτελέσει η GPU. Στην πρώτη υλοποίηση χρησιμοποιήθηκε μια διάσταση για τα threads και ένα μόνο μπλοκ από threads για λόγους εξοικείωσης με την βιβλιοθήκη και για απλότητα. Στην πρώτη υλοποίηση ο περιορισμός είναι 1024 threads δηλαδή 1024 νευρώνες ανά επίπεδο. Στις εφαρμογές που χρησιμοποιήθηκε, δεν χρειάστηκαν περισσότεροι από 256 νευρώνες.



Κάθε thread υλοποιεί την λειτουργία ενός νευρώνα :

$$f(x_i, w_i) = \phi\left(\sum_i (w_i \cdot x_i)\right)$$

Δηλαδή διαβάζει την πληροφορία εισόδου x_i και την πολαπλασιάζει με την τρέχουσα τιμή του βάρους του. Αν θεωρήσουμε ότι το **W** και το **X** είναι ένα διάνυσμα το άθροισμα που υπολογίζεται δεν είναι άλλο από το εσωτερικό γινόμενο των δυο διανυσμάτων. Εν συνέχεια, όπως είναι γνωστό το αποτέλεσμα φιλτράρεται από μια μη γραμμική συνάρτηση ϕ .

Για την παραλληλοποίηση προβλημάτων τις περισσότερες φορες γίνεται σε αναγωγή στις εξής κατηγορίες :

1. **Αντιστοίχιση (Map) ή 1 προς 1.** Κάθε επεξεργαστική μονάδα διαβάζει από διαφορετικές θέσεις μνήμης και γράφει σε διαφορετική θέση μνήμης. Αυτά είναι τα πιο εύκολα προβλήματα παραλληλισμού και συνήθως αναφερόμαστε σε αυτά ως embarrassing parallel. Τέτοιο πρόβλημα είναι η πρόσθεση 2 διανυσμάτων και η αποθήκευση τους σε ένα τρίτο.
2. **Συλλογή (Gather) ή αλλιώς πολλά σε 1.** Κάθε νήμα διαβάζει από διαφορετική θέση μνήμης, αλλά επιδιώκουν να γράψουν στην ίδια θέση μνήμης. Σε αυτή την περίπτωση δημιουργούνται προβλήματα ανταγωνισμού κοινής μνήμης.
3. **Διασκόρπιση(Scatter) ή αλλιώς 1 σε πολλά.** Ενώ διαβάζουν όλα από την ίδια θέση μνήμης κάθε νήμα διαλέγει πολλούς πιθανούς προορισμούς εγγραφής.
4. **Stencil (λίγα σε ένα).** Υποπερίπτωση της **Συλλογής(2)** μόνο που τα γειτονικά στοιχεία λαμβάνονται υπόψιν και γράφουν όλα σε μια κοινή θέση μνήμης.
5. **Ελάττωση (Reduce) ή όλα σε 1.** Λαμβάνονται όλα τα στοιχεία και το αποτέλεσμα είναι σε μια θέση μνήμης. Για παράδειγμα , η πρόσθεση όλων των στοιχείων ενός πίνακα και το αποτέλεσμα τους σε μια θέση μνήμης.
6. **Σκανάρισμα και ταξινόμηση(Scan & Sort) ή αλλιώς από όλα σε όλα.** Η πιο σύνθετη κατηγορία αναγωγής προβλημάτων. Όλα τα στοιχεία εισόδου επηρεάζουν την έξοδο. Απαιτούν ιδιαίτερη προσοχή στην επίλυση τέτοιων προβλημάτων, σε παράλληλο προγραμματισμό.

Μελετώντας καλύτερα την εμπρόσθια διάδοση σε ένα ΤΝΔ, παρατηρήθηκε ότι η μοντελοποίηση του νευρώνα ανάγεται σε ένα πρόβλημα Ελάττωσης(reduce). Διαβάζεται η τιμή όλων των εισόδων πολλαπλασιασμένη με το αντίστοιχο βάρος και το αποτέλεσμα αποθηκεύεται σε μια μόνο θέση μνήμης. Η μοναδική προσθήκη είναι η μη γραμμική συνάρτηση στο τελικό άθροισμα που δεν επηρεάζει την αναγωγή.

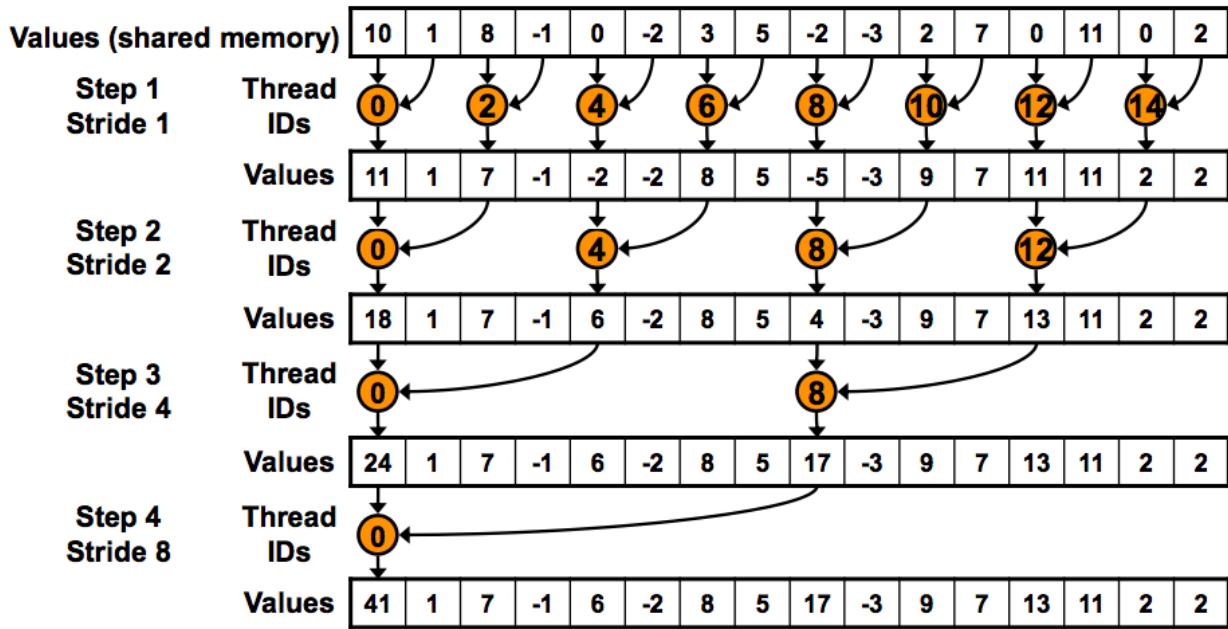
Προς αυτήν την κατεύθυνση, δημιουργήθηκε η βελτιστοποιημένη έκδοση για την εμπρόσθια διάδοση. Η ελάττωση λειτουργεί βασιζόμενη στην προσεταιριστική ιδιότητα της πρόσθεσης.

$$\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$$

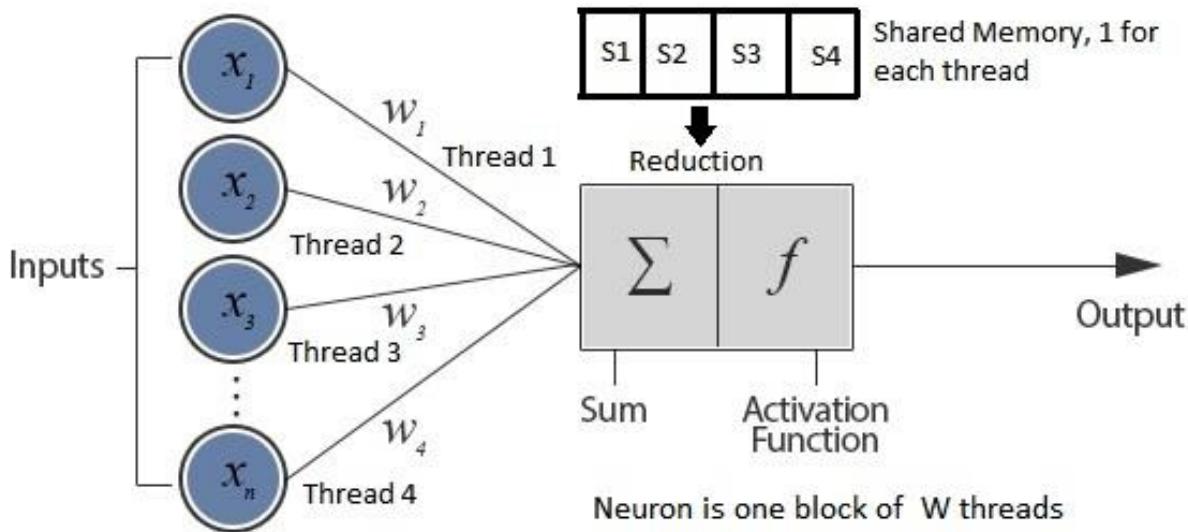
Ο παραπάνω τύπος γενικεύεται και για N όρους, Στο πλαίσιο αυτό πολλές επεξεργαστικές μονάδες αθροίζουν ανά δύο τους όρους και αποθηκεύουν το αποτέλεσμα στην κοινή μνήμη. Εν συνέχεια, παίρνουμε επαναληπτικά τα αποτελέσματα των αθροισμάτων ανά 2 μέχρι να καταλήξουμε να έχουμε το τελικό αποτέλεσμα. Ένα παράδειγμα παρουσιάζεται παρακάτω για τον υπολογισμό του αθροίσματος 16 όρων:



Βασική υλοποίηση του reduction



Με αυτήν την θεώρηση κάθε νευρώνας αποτελείται από πολλά threads, όσα και το διάνυσμα εισόδου του προηγούμενου επιπέδου. [Παράρτημα A] Τα thread υπολογίζουν το τοπικό γινόμενο του βάρους και της εισόδου και αποθηκεύουν το τοπικό αποτέλεσμα στην κοινή μνήμη. Τέλος στην κοινή μνήμη εκτελείται η ελάττωση. Κάθε νευρώνας τώρα είναι ένα μπλοκ από νήματα και μόνο μέσα σε αυτό το μπλοκ είναι ορατή για κάθε νευρώνα η κοινή μνήμη. Μια σχετική μοντελοποίηση παρουσιάζεται παρακάτω σχήμα:



Ένας σοβαρός περιορισμός είναι ότι για την εφαρμογή της ελάττωσης, οι N όροι του αθροίσματος (δηλαδή οι νευρώνες του προηγούμενου επιπέδου) πρέπει να είναι δύναμη του 2 ($N = 2^k$). Ένας



απλός τρόπος είναι να συμπληρώσουμε τους όρους του αθροίσματος με μηδενικά ώστε να είναι δύναμη του $2(N + X_0 = 2^k)$. Το overhead της υλοποίησης αυτής είναι ένα επιπλέον if-else statement στο GPU Kernel. Όσον αφορά την κοινή μνήμη που απαιτείται να διαθέτει η κάρτα γραφικών για το επίπεδο i είναι : $Shared\ Memory(i) = InputVector(i) \times sizeof(float) \times Neurons(i)$

Υπάρχουν πολλές δυνατές υλοποιήσεις του reduction[link]. Στόχος είναι η καλύτερη δυνατή αξιοποίηση των πόρων της GPU. Ενώ η βασική υλοποίηση του reduction είναι σχετική απλή, είναι δύσκολο να ληφθούν τα σωστά αποτελέσματα. Γι' αυτό με κάθε δοκιμή που πραγματοποιήθηκε υπολογιζόταν το αθροιστικό σφάλμα του διανύσματος εξόδου των νευρώνων ενός επιπέδου με την αρχική υλοποίηση που δεν χρησιμοποιήθηκε reduction.

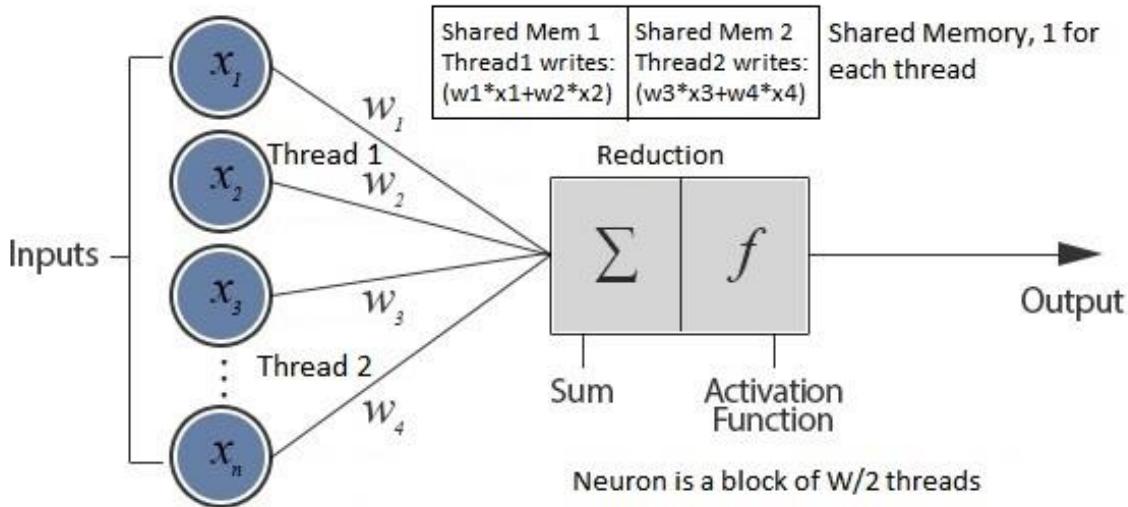
Η ελάττωση έχει πολύ χαμηλή αριθμητική πυκνότητα 1 πράξη ανά στοιχείο που φορτώνεται και συνεπώς πρέπει να προσπαθήσουμε για μέγιστο εύρος ζώνης. Οι βελτιστοποιήσεις εφαρμόζονται αυξητικά μέχρι την τελική έκδοση του Kernel. Αρχικά, γίνεται προσπάθεια προσθήκης όσο πιο κοντινών αθροισμάτων για ακολουθιακή πρόσβαση στην μνήμη. Σε επόμενο στάδιο βελτιστοποίησης πραγματώνεται η ελαχιστοποίηση των διακλαδώσεων και η υλοποίηση της πρώτης πρόσθεσης κατά την φόρτωση από την global μνήμη. Επειδή ξέρουμε τον αριθμό των επαναλήψεων (που είναι ίσος με το διάνυσμα εισόδου) την ώρα της μεταγλώττισης, μπορούμε να 'ξετυλίξουμε' τον βρόγχο των διαδοχικών προσθέσεων. Ως αποτέλεσμα, ο κώδικας σε GPU γίνεται όλο και πιο ταχύς.[Παράρτημα A]

Στην υλοποίηση που έχει αναλυθεί μέχρι τώρα, **κάθε νήμα ήταν ένα βάρος**. Αυτή η τεχνική μας ωθεί να χρησιμοποιήσουμε μεγάλο αριθμό νημάτων, που συνήθως είναι καλό. Το πρόβλημα είναι ότι κάθε νήμα υπολογίζει μόνο ένα γινόμενο και αποθηκεύει το αποτέλεσμα στην κοινή μνήμη. Όταν το διάνυσμα εισόδου είναι αρκετά μεγάλο αποθηκεύουμε επίσης και αρκετή μνήμη.

Μια ιδέα λοιπόν είναι κάθε νήμα να υπολογίζει δύο εσωτερικά γινόμενα. Δηλαδή, κατά την φόρτωση από την global μνήμη να υπολογίζει το άθροισμα των 2 γινομένων και να το αποθηκεύει στην διαμοιραζόμενη μνήμη. Έτσι το reduction θα χρειαστεί τις μισές προσθέσεις για να παρθεί το αποτέλεσμα όπως επίσης και την μισή κοινή μνήμη. Για αρκετά μεγάλο διάνυσμα εισόδου κάτι τέτοιο μπορεί να αυξήσει τρομακτικά την επιτάχυνση. Βέβαια για την υλοποίηση χρειάστηκε αρκετή προσπάθεια για να διατηρηθεί η γενικότητα. Επειδή απαιτούνται τα μισά νήματα η λύση αυτή περιορίζεται από τον αριθμό των εισόδων να είναι άρτιος αριθμός και μεγαλύτερος του $N_{critical}$. Για την GPU που χρησιμοποιήθηκε (Nvidia GeForce 920M) $N_{critical} = 128$. Για μικρότερο αριθμό νημάτων τα αποτελέσματα δεν είχαν σημαντική βελτίωση.



Σχηματική αναπαράσταση:

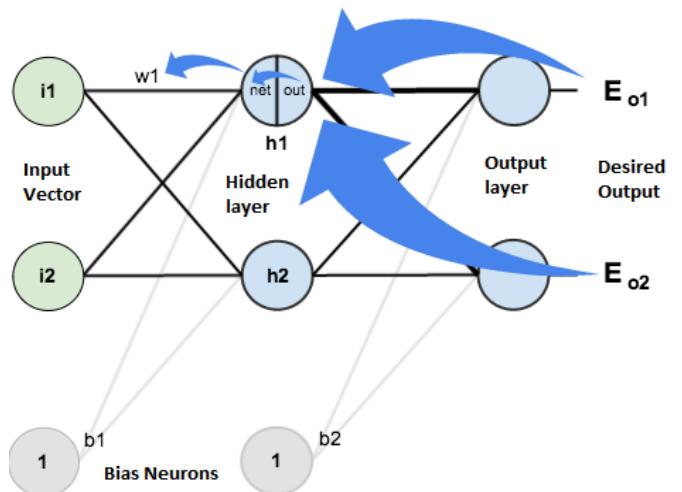


Υλοποίηση BPA και δοκιμές βελτιστοποίησης

Η αρχική υλοποίηση του BPA σε GPU έγινε αντιστοιχίζοντας **κάθε νήμα σε έναν νευρώνα**. Οι απαιτήσεις μνήμης στον BPA είναι σαφώς μεγαλύτερες καθώς χρειάζονται πληροφορίες από το επόμενο επίπεδο. Κάθε επίπεδο χαρακτηρίζεται, σε μια επανάληψη κατά την εκπαίδευση, από τα εξής:

1. Διάνυσμα εισόδου
2. Συντελεστές βαρύτητας επιπέδου
3. Διάνυσμα εξόδου
4. Διάνυσμα συντελεστών δέλτα
5. Ρυθμός εκπαίδευσης (κοινός για όλο το ΤΝΔ)

Στον BPA, είτε για το τελευταίο επίπεδο, είτε για τα κρυφά, απαιτείται η πληροφορία του επόμενου επιπέδου. Με αυτόν τον τρόπο το σφάλμα διαδίδεται δυναμικά από την έξοδο προς την είσοδο. Προφανώς το επίπεδο εξόδου, αντί για το επόμενο επίπεδο λαμβάνει υπόψιν το διάνυσμα των επιθυμητών τιμών για το κάθε παράδειγμα εκπαίδευσης. Η ειδοποιός διαφορά έγκειται στον υπολογισμό των συντελεστών Δέλτα του τρέχοντος επιπέδου με την πληροφορία του επομένου, ως ένα εσωτερικό γινόμενο των Δέλτα του επόμενου επιπέδου με τα σχετικά βάρη. Στην διπλανή εικόνα φαίνεται μια οπτική αναπαράσταση.



Εφόσον έχει πραγματοποιηθεί ο υπολογισμός των Δέλτα για κάθε νευρώνα, κάθε νήμα αλλάζει τα β άρη σύμφωνα με την σχέση $\Delta w(i) = -\Delta \text{έλτα} \times \text{Είσοδος}(i) \times \text{Συντελεστής}_i \cdot \text{Εκπαίδευσης}$ [Παράρτημα A]

Δοκιμάστηκε για τους νευρώνες εξόδου να οριστεί ένα μπλοκ ανά νευρώνα με νήματα όσα τα βάρη. Ωστόσο, δεν μετρήθηκε επιτάχυνση σε σχέση με την προηγούμενη υλοποίηση.

Υλοποίηση παραλλαγών BPA

Πριν τις υλοποίησεις κρίνεται μείζονος σημασίας να αναλυθεί ο αριθμός των συντελεστών βαρύτητας για ένα TNΔ, διθέντος Ν επιπέδων και k_i νευρώνων ανά επίπεδο με k_0 το μέγεθος του διανύσματος εισόδου.

$$\text{αριθμός βαρών} = \sum_1^N k_i * k_{i-1}$$

Επομένως κάθε φορά που προστίθεται μια επιπλέον πληροφορία για κάθε βάρος, όπως θα δειχθεί παρακάτω, οι απαιτήσεις μνήμης αυξάνονται ανάλογα. Γι' αυτό είναι καλή τεχνική να υπάρχει μια συνάρτηση που θα υπολογίζει των αριθμών των συντελεστών του δικτύου για πολλούς λόγους, για τις διάφορες τοπολογίες που δοκιμάζονται. Αρχικά, μας δίνει μια αίσθηση της μνήμης που απαιτείται

$$\text{μνήμη} = \text{αριθμός βαρών} * \text{μέγεθος μεταβλητής}$$

Για προγραμματισμό σε GPU πρέπει να έχουμε πάντα μια αίσθηση της μνήμης που χρησιμοποιούμε διότι αφενός είναι περιορισμένη (συνήθως 2 GB global memory για έναν προσωπικό υπολογιστή), και αφετέρου στόχος μας είναι να ελαχιστοποιήσουμε της μεταφορές μεταξύ μνήμης CPU και GPU. Επειδή τα βάρη καθορίζουν την μοντελοποίηση του συστήματος αναγνώρισης για πεπερασμένο αριθμό δεδομένων, ενδείκνυται ο αριθμός των συντελεστών w να είναι τουλάχιστον μιας τάξης μεγέθους λιγότερη. Περισσότερες πληροφορίες στην παράγραφο Underfitting & Overfitting. Τέλος επισημαίνεται ότι οι παραλλαγές επηρεάζουν μόνο την οπισθοδρομική διάδοση του σφάλματος, η εμπρόσθια διάδοση παραμένει αναλλοίωτη.

Momentum:

Η υλοποίηση της παραλλαγής αυτής απαιτεί την αποθήκευση για κάθε συντελεστή την προηγούμενη μεταβολή του βάρους του $\Delta W_{previous}$. Απαιτείται δέσμευση χώρου σε CPU καθώς και GPU κατά την αρχικοποίηση του προγράμματος. Η αριθμητική τιμή της αδράνειας αρχικοποιείται στην βιβλιοθήκη με την τιμή 0.6. Η τιμή προκύπτει από πειραματικές μελέτες που έχουν διεξαχθεί σε διάφορα σύνολα δεδομένων. Έτσι ο χρήστης της βιβλιοθήκης χρειάζεται να προσδιορίσει μόνο την παράμετρο του συντελεστή εκπαίδευσης. Κάθε Kernel του BPA λαμβάνει μια επιπλέον δομή των $\Delta W_{previous}$ του τρέχοντος επιπέδου. Αφότου υπολογιστεί η νέα μεταβολή ΔW η τιμή της προηγούμενης επανάληψης αντικαθίσταται με την νέα για την επόμενη επανάληψη. Είναι φανερό πως αυτή η παραλλαγή δεν απαιτεί μεγάλη προγραμματιστική πολυπλοκότητα.

Quick propagation:

Εδώ δεν αρκεί η αποθήκευση του $\Delta W_{previous}$ αλλά χρειάζεται και η αποθήκευση της τοπικής κλίσης για κάθε βάρος $Gradient_{previous} = \delta_{neuron} * O_j$ όπου O_j η προηγούμενη έξοδος του νευρώνα j . Εκτός



από τις απαιτήσεις σε μνήμη η προσέγγιση της δεύτερης παραγώγου εισάγει νέα προβλήματα. Υπενθυμίζεται για ευκολία του αναγνώστη ότι τα w αλλάζουν ως εξής :

$$\Delta^{(k)} w_i = \Delta^{(k-1)} w_i \left(\frac{\nabla_i E^{(k)}}{\nabla_i E^{(k-1)} - \nabla_i E^{(k)}} \right)$$

Τα προβλήματα αυτής της μεθόδου γίνονται αντιληπτά κατά την υλοποίηση. Ένα υπολογιστικό σύστημα χρησιμοποιεί συγκεκριμένο αριθμό bits για την ακρίβεια των αριθμητικών πράξεων. Λόγω του παρανομαστή μπορεί να προκύψει μη προβλεπόμενη συμπεριφορά καθώς η διαφορά των κλίσεων μεταξύ των διαδοχικών επαναλήψεων γίνεται ιδιαίτερα μικρή. Για την αντιμετώπιση αυτού του προβλήματος έχουν προταθεί διάφορες μέθοδοι [\[link\]](#). Μια απλή προσέγγιση είναι να ελέγχεται η απόλυτη τιμή της διαφοράς να μην ξεπερνά ένα συγκεκριμένο κατώφλι, δεδομένης της ακρίβειας. Ιδανικά θα θέλαμε τότε να σταματάει η εκπαίδευση. Ως αποτέλεσμα τίθενται νέοι περιορισμοί ως προς τον μέγιστο αριθμό επαναλήψεων στις οποίες μπορεί να πραγματωθεί εκπαίδευση.

Resilient:

Για τον επαναπροσδιορισμό των νέων βαρών τώρα χρειαζόμαστε

- Την προηγούμενη μεταβολή των βαρών $\Delta W_{previous}$
- Την προηγούμενη κλίση κάθε βάρους $Gradient_{previous}$
- Την νέα μεταβλητή που επηρεάζει το μέτρο αλλαγής των βαρών Dij

Για να γίνει κατανοητή η απαίτηση μνήμης ας μελετήσουμε ένα μικρό δίκτυο με 19 δεδομένα εισόδου , 2 επίπεδα με 30 και 10 νευρώνες αντίστοιχα. Ο συνολικός αριθμός συντελεστών είναι, μαζί με bias νευρώνες, 910. Αν κάθε στοιχείο είναι τύπου float και αντιστοιχεί σε 4 byte ο κλασσικός αλγόριθμος χρειάζεται 3.55 Mbyte, ενώ ο Resilient 14.22 Mbyte.

Αφού δεσμεύσουμε όλη την απαραίτητη μνήμη, χρειάζεται να περάσουμε σαν παράμετρο τις πληροφορίες αυτές για το τρέχον επίπεδο εκπαίδευσης κατά την εκτέλεση του πυρήνα. Η παραλλαγή του αλγορίθμου έγινε όπως ακριβώς περιγράφετε στον ψευδοκώδικα [\[link\]](#).

$$\frac{\partial E^{(t-1)}}{\partial w_{ij}} * \frac{\partial E^{(t)}}{\partial w_{ij}}$$

Μεγάλη προσοχή χρειάζεται κατά τον υπολογισμό του γινομένου :

Δοθέντος της περιορισμένης ακρίβειας απαιτείται ειδική μεταχείριση κατά τον υπολογισμό του γινομένου. Για την συγκεκριμένη περίπτωση δεν μας ενδιαφέρει η ακριβής αριθμητική τιμή αλλά το πρόσημο. Χρησιμοποιείται πάλι ένα κατώφλι κάτω από το οποίο το γινόμενο θεωρείται 0, αλλιώς επιστρέφεται μια θετική τιμή αν το γινόμενο είναι θετικό και μεγαλύτερο του κατωφλίου. Όμοια αντιμετώπιση λαμβάνεται και για τις αρνητικές τιμές. Τέτοια λογικά λάθη προκύπτουν συχνά σε προγραμματισμό σε GPU και είναι δύσκολα ανιχνεύσιμα. Οι παράμετροι η^+ , η^- αρχικοποιούνται με τις πλέον συνήθεις τιμές 1.2 και 0.5 αντίστοιχα. Εφόσον η σύγκλιση γίνεται ταχύτερα, ο αριθμός των επαναλήψεων πρέπει να μειωθεί.



5. Εκπαίδευση και Αξιολόγηση ΤΝΔ

Περιγραφή Datasets

Παρακάτω γίνεται μια σύντομη αναφορά στα σύνολα δεδομένων που δοκιμάστηκε ταξινόμηση με ΤΝΔ. Να σημειωθεί ότι ο επιστήμονας που μελετάει σύνολα δεδομένων δεν χρειάζεται να έχει γνώση των δεδομένων που αναπαρίστανται. Ούτε αν επιτευχθεί αναγνώριση θα έχει κατανοήσει τον τρόπο που οι μη γραμμικοί νευρώνες μοντελοποιούν την είσοδο.

Σύνολο δεδομένων από οίνους

Τα στοιχεία που χρησιμοποιήθηκαν είναι τα αποτελέσματα μιας χημικής ανάλυσης των οίνων που καλλιεργούνται στην ίδια περιοχή της Ιταλίας, αλλά παράγονται από τρεις διαφορετικές ποικιλίες σταφυλιού. Η ανάλυση καθόρισε τις ποσότητες των 13 συστατικών που βρέθηκαν σε καθέναν από τους τρεις τύπους οίνων. Σε επίπεδο ταξινόμησης, είναι ένα καλά δημιουργημένο πρόβλημα. Έχει οργανωμένες δομές. Ενδείκνυται σαν καλό σύνολο δεδομένων για την πρώτη δοκιμή ενός νέου ταξινομητή, αλλά όχι πολύ δύσκολο. Μερικά χημικά χαρακτηριστικά είναι :

- | | |
|------------------------|--------------------------------------|
| 1) Αλκοόλ | 8) Μη φλαβονοειδείς φαινόλες |
| 2) Μηλικό οξύ | 9) Προανθοκυανίνες |
| 3) Τέφρα | 10) Ένταση χρώματος |
| 4) Αλκαλικότητα τέφρας | 11) Απόχρωση |
| 5) Μαγνήσιο | 12) OD280/OD315 των αραιωμένων οίνων |
| 6) Ολικές φαινόλες | 13) Προλί |
| 7) Φλαβονοειδή | |

Δεδομένα καρκίνου του μαστού

Τα χαρακτηριστικά υπολογίζονται από μια ψηφιοποιημένη εικόνα ενός λεπτού αναρροφήματος βελόνας (FNA) μιας μάζας στήθους. Περιγράφουν τα χαρακτηριστικά των κυτταρικών πυρήνων που υπάρχουν στην εικόνα. Ο πρώτος διαχωρισμός των δεδομένων έγινε με χρήση μεθόδου πολλαπλών επιφανειών, μια μέθοδος η οποία χρησιμοποιεί γραμμικό προγραμματισμό για την κατασκευή ενός δέντρου αποφάσεων. Επισημαίνονται τα βασικά χαρακτηριστικά :

- | | |
|--|--|
| • ταυτότητα ασθενή (ID) | • ομαλότητα (τοπική μεταβολή στο μήκος ακτίνας) |
| • Διάγνωση (M = κακοήθη, B = καλοήθη) | • πυκνότητα (περίμετρο \wedge^2 / περιοχή - 1,0) |
| • Για κάθε πυρήνα κυττάρων υπολογίζονται δέκα χαρακτηριστικά : | • κοίλωμα (σοβαρότητα κοίλων τμημάτων του περιγράμματος) |
| • ακτίνα (μέση απόσταση από κέντρο σε σημεία στην περίμετρο) | • κοίλα σημεία (αριθμός κοίλων τμημάτων του περιγράμματος) |
| • υφή (τυπική απόκλιση τιμών γκρι κλίμακας) | • fractal διάσταση ("προσέγγιση ακτογραμμής" - 1 |
| • περίμετρος | |
| • περιοχή | |



Σύνολο από παρτίδες του παιχνιδιού πόκερ

Κάθε καταγραφή είναι ένα παράδειγμα ενός χεριού πόκερ, που αποτελείται από πέντε κάρτες που προέρχονται από μια γνωστή τράπουλα των 52 φύλλων. Κάθε κάρτα περιγράφεται με δύο χαρακτηριστικά (σχήμα και κατάταξη), δηλαδή το σύνολο αποτελείται από 10 προγνωστικά χαρακτηριστικά. Υπάρχει μια κλάση την οποία θέλουμε να αναγνωρίσουμε που περιγράφει την δύναμη του χεριού ανάμεσα σε άλλα(ένα ζευγάρι , δύο ζευγάρια κτλ.).

Magic Gamma Telescope Data Set

Τα δεδομένα δημιουργούνται για την προσομοίωση της εγγραφής σωματιδίων γάμμα υψηλής ενέργειας σε ένα τηλεσκόπιο Cherenkov χρησιμοποιώντας τη τεχνική απεικόνισης. Το τηλεσκόπιο Cherenkov παρατηρεί υψηλής ακτινοβολίας ακτίνες γάμμα, εκμεταλλεύμενο την ακτινοβολία που εκπέμπεται από τα φορτισμένα σωματίδια που παράγονται μέσα στην ατμόσφαιρα. Αυτή η ακτινοβολία (από ορατά έως UV μήκη κύματος) διαρρέει μέσω της ατμόσφαιρας και καταγράφεται στον ανιχνευτή, επιτρέποντας την ανακατασκευή των παραμέτρων τους. Οι διαθέσιμες πληροφορίες συνίστανται από παλμούς που αφήνονται από τα εισερχόμενα φωτόνια στους σωλήνες φωτοπολλασιασμού, τοποθετημένοι σε ένα επίπεδο. Ανάλογα με την ενέργεια της πρωτογενούς ακτινοβολίας γάμμα, συλλέγονται συνολικά μερικές εκατοντάδες έως περίπου 10.000 φωτόνια, σε πρότυπα , επιτρέποντας στατιστικά να διακρίνουν εκείνα που προκαλούνται από **πρωτογενή γάμμα** ή από **κοσμικές ακτίνες** στην ανώτερη ατμόσφαιρα . Οι τελευταίες είναι και οι κλάσεις στις οποίες γίνεται προσπάθεια αναγνώρισης. Το σετ περιέχει 19000 δεδομένα με 11 πραγματικές τιμές το καθένα.

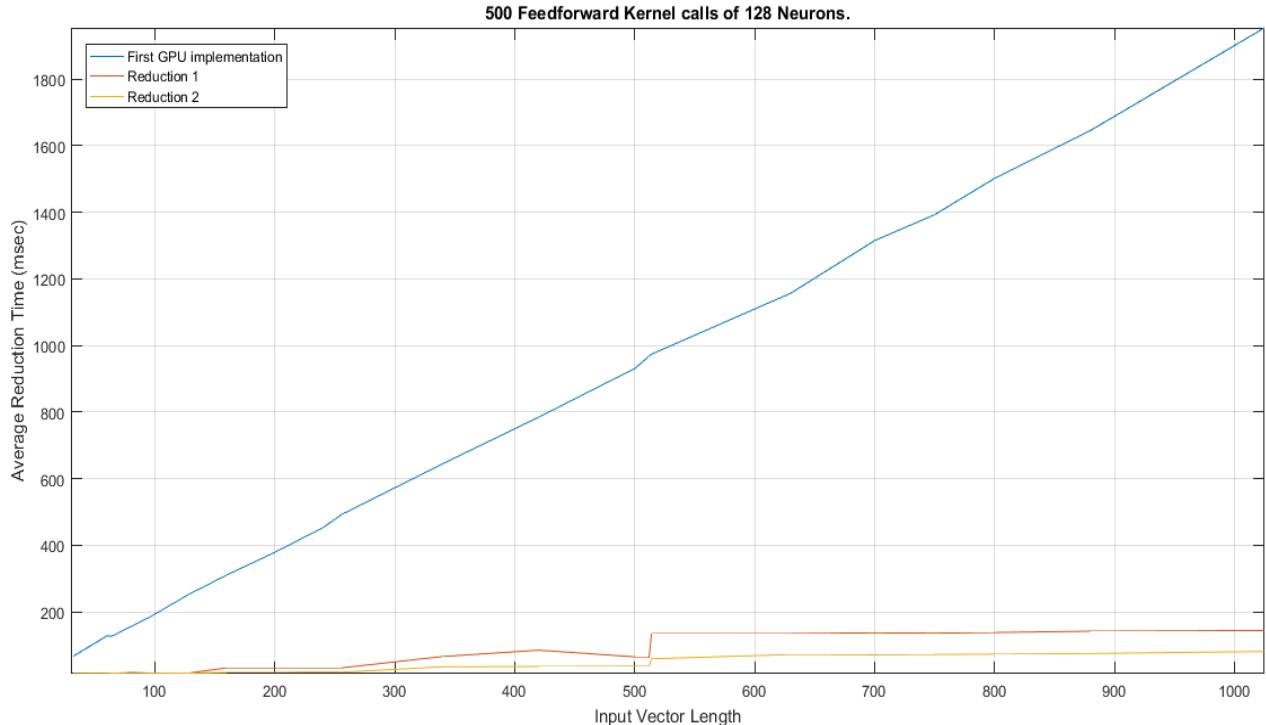


6. Αποτελέσματα και συμπεράσματα

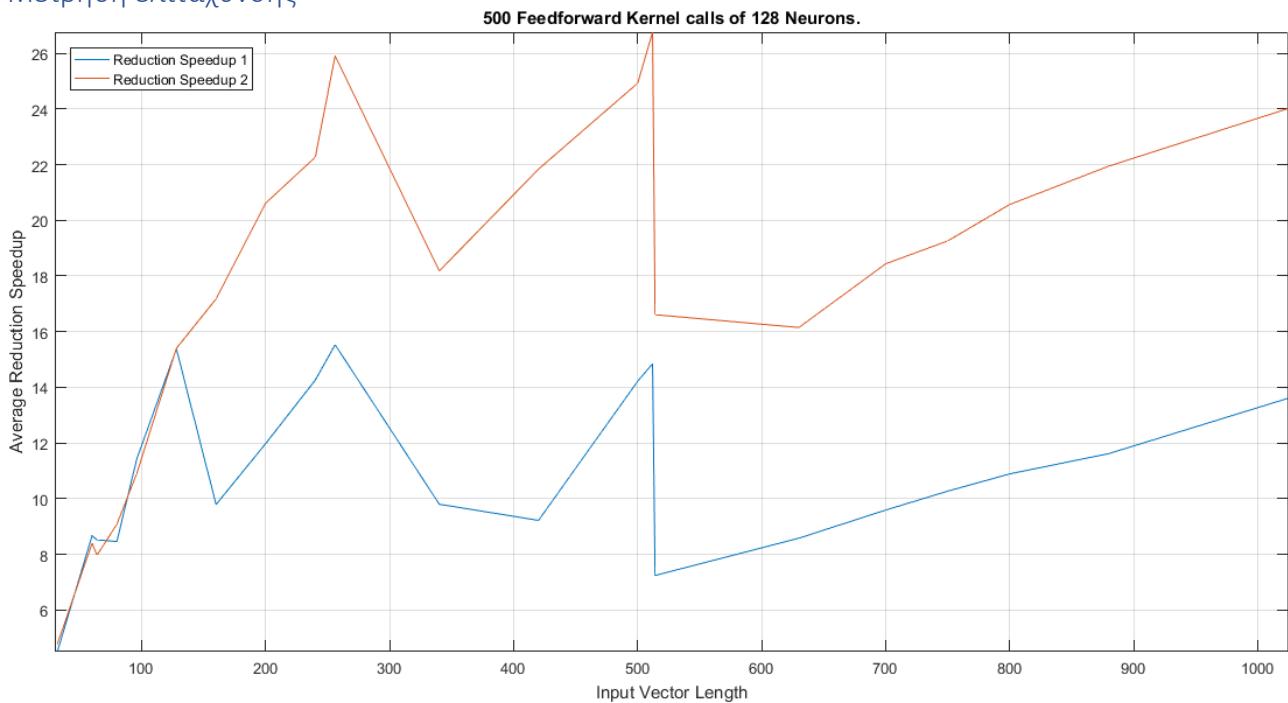
Δοκιμές και βελτιστοποιήσεις

Παρακάτω παρουσιάζονται τα αποτελέσματα από την εφαρμογή του reduction για ένα επίπεδο με διαφορετικά μεγέθη διανύσματος εισόδου. Επιπλέον γίνεται σύγκριση με την αρχική υλοποίηση σε GPU.

Μέτρηση χρόνου



Μέτρηση επιτάχυνσης



Σχολιασμός αποτελεσμάτων reduction

Παρατηρήθηκε με πειραματικές δοκιμές, ότι η αναγωγή των όρων στην επόμενη δύναμη του 2 επηρεάζει την επιτάχυνση του αλγορίθμου. Όσο πιο πολύ απέχει ο αριθμός του διανύσματος εισόδου από την επόμενη δύναμη του 2, τόσο χειρότερο είναι το speedup. αυτό συμβαίνει λόγο του αυξημένου divergence στο if-else, όπως επίσης και λόγω των περιττών προσθέσεων σε σχέση με την πρώτη υλοποίηση.

Για παράδειγμα για ένα διάνυσμα εισόδου με 520 στοιχεία η αρχική υλοποίηση υπολογίζει τα 520 εσωτερικά γινόμενα και τα αθροίζει 520 φορές σειριακά στον ένα νευρώνα. Αντίθετα με την υλοποίηση του reduction ο αλγόριθμος επεκτείνει το διάνυσμα σε μήκος 1024, που είναι η επόμενη δύναμη του 2. Συνεπώς εκτελεί 1024 προσθέσεις παράλληλα στον έναν νευρώνα(504 παραπάνω πράξεις).

Γι' αυτό παρατηρείται σε κάθε δύναμη του 2 του διανύσματος εισόδου τοπικό μέγιστο στην επιτάχυνση. Βλέπουμε ακόμη πως υπάρχει επιτάχυνση και για σχετικά μικρό αριθμό διανύσματος εισόδου. Επισημαίνεται ξανά ότι η είσοδος για το πρώτο επίπεδο είναι το δεδομένο εκπαίδευσης ενώ για τα υπόλοιπα επίπεδα η είσοδος του προηγούμενου επιπέδου. Παρατηρήθηκε ότι η επιτάχυνση δεν επηρεάζεται από τους νευρώνες του τρέχοντος επιπέδου. Τέλος ο χρόνους του σειριακού υπολογισμού αυξάνεται γραμμικά που δημιουργεί μεγάλη καθυστέρηση σε μεγάλα μοντέλα. Η μέγιστη επιτάχυνση που μετρήθηκε σε σχέση με την αρχική έκδοση ήταν 26,7.

Σύγκριση αποτελεσμάτων

Αποτελέσματα εκπαίδευσης και επαλήθευσης με το training set



| Wine Dataset | Breast cancer dataset | Poker rules learning |
|----------------------------------|----------------------------------|-----------------------------------|
| 178 instances | 569 instances | 25010 instances |
| 275 weights | 398 weights | 1795 weights |
| Max epochs = 12000 | Max epochs = 15000 | Max epochs = 50000 |
| Level 1 with 16 neurons | Level 1 with 12 neurons | Level 1 with 128 neurons |
| Level 2 with 3 neurons | Level 2 with 2 neurons | Level 2 with 4 neurons |
| Average training iterations 2822 | Average training iterations 6296 | Average training iterations 30000 |
| Min error = 0.0001 | Min error = 0.0004 | Min error = 0.0002 |
| 99.4 success percentage | 98.06 success percentage | 50.45 success percentage |
| TIME = 1 SECOND | TIME = 2.5 SECONDS | TIME = 12.3 SECONDS |



Σχολιασμός αποτελεσμάτων από την εκπαίδευση και αξιολόγηση με το training set

Αρχικά, επιβεβαιώνεται η σύγκλιση του αλγορίθμου. Δοκιμάστηκε ο κλασσικός αλγόριθμος BPA. Τα αποτελέσματα είναι περισσότερο για να εξεταστεί ότι το TNΔ μπορεί να μοντελοποιήσει την έξοδο ικανοποιητικά καλά, κάτι το οποίο δεν κατάφερε να επιτευχθεί στο σύνολο δεδομένων από παρτίδες πόκερ. Αυτό μπορεί να συμβεί για πολλούς λόγους όπως μεγάλη μη γραμμικότητα μοντέλου, πολλές κλάσεις με ανομοιόμορφη κατανομή, λάθος επιλογή αλγόριθμου εκπαίδευσης. Παρόλα αυτά δεν εξασφαλίζεται σε καμία περίπτωση γενίκευση, για να φτάσει σε τόσο υψηλά ποσοστά προφανώς υφίσταται overfitting. Θα εξεταστούν διάφορες τεχνικές για την αποφυγή του τελευταίου.

Αποτελέσματα εκπαίδευσης και αξιολόγησης με διαφορετικά δεδομένα

//TODO



Μελλοντικές βελτιώσεις

Η τρέχουσα έκδοση της βιβλιοθήκης TNΔ υποστηρίζει την τοπολογία του πλήρους διασυνδεδεμένου δικτύου γι' αυτό και η χρησιμότητα της είναι περιορισμένη. Επίσης δεν παρέχεται κάποια αυτοματοποιημένη διαδικασία για την επίλυση πρακτικών ζητημάτων όπως το διάβασμα των δεδομένων και η αποθήκευση σου σε κατάλληλη δομή, όπως επίσης και ο διαχωρισμός του συνόλου δεδομένων για εκπαίδευση και αξιολόγηση. Μερικές από τις δυνατές μελλοντικές βελτιώσεις θα ήταν η δημιουργία ενός interface που να αυτοματοποιεί αυτές τις διαδικασίες για τον χρήστη. Επίσης, μελλοντικά θα διερευνηθεί και η δυνατότητα δημιουργίας πιο πολύπλοκων τοπολογιών υλοποιημένα σε GPU, όπως τα συνελικτικά νευρωνικά δίκτυα και ο συνδυασμός τους για την δημιουργία μοντέλων βαθιάς μάθησης(deep learning).

++



Παράρτημα Α

Εγκατάσταση του Cuda toolkit σε Windows

Προαπαιτούμενα:

Για να γίνει με επιτυχία η εγκατάσταση πρέπει απαραίτητα ο υπολογιστής να έχει :

- Μια κάρτα γραφικών της Nvidia που να υποστηρίζει Cuda [0]
- Μια υποστηριζόμενη έκδοση Microsoft Windows
- Μια υποστηριζόμενη έκδοση Microsoft Visual Studio

Για την εγκατάσταση σε λειτουργικό σύστημα Windows προτείνεται να έχετε Windows 7 , 8.1 ή 10 . Αρχικά , πρέπει να κατεβάσετε από την ιστοσελίδα της Nvidia [1] την βιβλιοθήκη ανάλογα με την αρχιτεκτονική του υπολογιστή σας. Υπενθυμίζεται ότι x86 είναι 32 μπit αρχιτεκτονική συστήματος , ενώ x86_64 είναι 64 μπit .

Table 1. Windows Operating System Support in CUDA 8.0

| Operating System | Native x86_64 | Cross (x86_32 on x86_64) |
|------------------|---------------|--------------------------|
| Windows 10 | YES | YES |
| Windows 8.1 | YES | YES |
| Windows 7 | YES | YES |

Table 2. Windows Compiler Support in CUDA 8.0

| Compiler | IDE | Native x86_64 | Cross (x86_32 on x86_64) |
|-----------------|------------------------------|---------------|--------------------------|
| Visual C++ 14.0 | Visual Studio 2015 | YES | YES |
| | Visual Studio Community 2015 | YES | NO |
| Visual C++ 12.0 | Visual Studio 2013 | YES | YES |
| Visual C++ 11.0 | Visual Studio 2012 | YES | YES |

Συμφώνα με τους πίνακες 1 και 2 (για την νεότερη έκδοση του Cuda που είναι μέχρι τώρα το 8.0) πρέπει να έχετε εγκαταστήσει το Visual Studio στις εκδόσεις που δείχνονται [2].

Κατά προτίμηση βάση των σχολίων από τους χρήστες δουλεύει πιο σταθερά σε αρχιτεκτονική x86_64 .

Αν θέλετε να δείτε την κάρτα γραφικών που έχει ο υπολογιστής σας τρέξτε στο τερματικό σας την εντολή: `control /name Microsoft.DeviceManager`

Αν ακολουθήσατε όλα τα προηγούμενα κάντε την εγκατάσταση του Cuda toolkit.

Προσοχή αν έχετε Windows 10 θα ήταν καλύτερο να απενεργοποιήσετε τα updates.

Αφού ολοκληρωθεί η εγκατάσταση μπορείτε να την επιβεβαιώσετε τρέχοντας στο command prompt την εντολή : `nvcc -V` Αν έχει γίνει σωστά η εγκατάσταση θα δείτε στην οθόνη του τερματικού σας την έκδοση του Cuda και του nvcc που είναι ο μεταγλωττιστής. Μαζί με την εγκατάσταση της βιβλιοθήκης εγκαθίστανται στον υπολογιστή και τα παραδείγματα για εξοικείωση που θα τα βρείτε κατά πάσα πιθανότητα στην τοποθεσία

`C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0\bin`

Δοκιμάστε να τρέξετε το πρόγραμμα deviceQuery που είναι ήδη μεταγλωτισμένο. Σε περίπτωση που δεν είναι μεταγλωτισμένο ανοίξτε τον κώδικα και κάντε compile μέσα από το Visual



Studio. Ανοίξτε από τον φάκελο εγκατάστασης το αρχείο project που είναι το visual studio (C:\ProgramData\NVIDIA Corporation\CUDA Samples\v8.0\1_Utils\deviceQuery)

Αν όλα πάνε καλά το αποτέλεσμα που θα δείτε θα είναι κάτι τέτοιο

```
deviceQuery.exe Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)
Detected 1 CUDA Capable device(s)

Device 0: "GeForce 920M"
  CUDA Driver Version / Runtime Version      8.0 / 8.0
  CUDA Capability Major/Minor version number: 3.5
  Total amount of global memory:             2048 MBytes (2147483648 bytes)
  ( 2 ) Multiprocessors, (192) CUDA Cores/MP:
    GPU Max Clock rate:                    954 MHz (0.95 GHz)
    Memory Clock rate:                     900 Mhz
    Memory Bus Width:                      64-bit
    L2 Cache Size:                         524288 bytes
  Maximum Texture Dimension Size (x,y,z)     1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
  Maximum Layered 1D Texture Size, (num) layers 1D=(16384), 2048 layers
  Maximum Layered 2D Texture Size, (num) layers 2D=(16384, 16384), 2048 layers
  Total amount of constant memory:            65536 bytes
  Total amount of shared memory per block:   49152 bytes
  Total number of registers available per block: 65536
  Warp size:                                32
  Maximum number of threads per multiprocessor: 2048
  Maximum number of threads per block:        1024
  Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
  Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
  Maximum memory pitch:                      2147483647 bytes
  Texture alignment:                          512 bytes
  Concurrent copy and kernel execution:       Yes with 1 copy engine(s)
  Run time limit on kernels:                 Yes
  Integrated GPU sharing Host Memory:        No
  Support host page-locked memory mapping:   Yes
  Alignment requirement for Surfaces:        Yes
  Device has ECC support:                   Disabled
  CUDA Device Driver Mode (TCC or WDDM):    WDDM (Windows Display Driver Model)
  Device supports Unified Addressing (UVA): Yes
  Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
  Compute Mode:
    < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

deviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1,
Force 920M
Result = PASS
```

Εγκατάσταση του Cuda toolkit σε Linux (Ubuntu μόνο έχει δοκιμαστεί)

Ελάχιστα προαπαιτούμενα για την εγκατάσταση:

- Μια κάρτα γραφικών της Nvidia που να υποστηρίζει Cuda
- Μια υποστηριζόμενη έκδοση Linux με τον gcc compiler
- Την βιβλιοθήκη Nvidia Cuda (deb αρχείο για Ubuntu διανομή)

Ο πίνακας 1 που ακολουθεί δείχνει περισσότερες πληροφορίες



Table 1. Native Linux Distribution Support in CUDA 8.0

| Distribution | Kernel | GCC | GLIBC | ICC | PGI | XLC | CLANG |
|-----------------|---------|-------|-------|----------------|-------------|--------|-------|
| x86_64 | | | | | | | |
| RHEL 7.x | 3.10 | 4.8.2 | 2.17 | 15.0.4 16.0 | 16.3+ NO | 3.8+ | |
| RHEL 6.x | 2.6.32 | 4.4.7 | 2.12 | | | | |
| CentOS 7.x | 3.10 | 4.8.2 | 2.17 | | | | |
| CentOS 6.x | 2.6.32 | 4.4.7 | 2.12 | | | | |
| Fedora 23 | 4.2.3 | 5.3.1 | 2.22 | | | | |
| OpenSUSE 13.2 | 3.16.6 | 4.8.3 | 2.19 | | | | |
| SLES 12 | 3.12.28 | 4.8.6 | 2.19 | | | | |
| SLES 11 SP4 | 3.0.101 | 4.3.4 | 2.11 | | | | |
| Ubuntu 16.04 | 4.4.0 | 5.3.1 | 2.23 | | | | |
| Ubuntu 14.04 | 3.13 | 4.8.2 | 2.19 | | | | |
| ARMv8 (aarch64) | | | | | | | |
| Ubuntu 14.04 | 3.13 | 4.8.2 | 2.19 | NO | NO | NO | NO |
| POWER8(*) | | | | | | | |
| RHEL 7.x | 3.10 | 4.8.2 | 2.17 | NO | NO | 13.1.2 | NO |
| Ubuntu 16.04 | 4.4.0 | 5.3.1 | 2.23 | NO | NO | 13.1.2 | NO |

Να σημειωθεί ότι η εγκατάσταση σε Linux απευθύνεται σε εξοικειωμένους χρήστες με το περιβάλλον του Linux και την μεταγλώττιση προγραμμάτων από την γραμμή εντολών. Δεν χρειάζεται να έχετε εμπειρία με παράλληλο προγραμματισμό ή με το Cuda. Επίσης πολλές εντολές μπορεί να χρειάζονται δικαιώματα διαχειριστή (root) για να εκτελεστούν.

Για να δείτε την nvidia κάρτα γραφικών σας μπορείτε να χρησιμοποιήσετε την εντολή :

```
$ lspci | grep -i nvidia
```

Για να δείτε την ακριβής έκδοση του λειτουργικού σας χρησιμοποιήστε την εντολή:

```
$ uname -m && cat /etc/*release
```

Για να δείτε την ακριβής έκδοση του gcc compiler σας χρησιμοποιήστε την εντολή:

```
$ gcc --version
```

Αν είστε σε λειτουργικό Ubuntu δοκιμάστε να βάλετε τα header files που χρειάζονται με την εντολή : `$ sudo apt-get install linux-headers-$(uname -r)`

Απεγκαταστήστε απαραίτητα όλους τους NVidia drivers ή οποιαδήποτε έκδοση της βιβλιοθήκης Cuda έχετε εγκαταστήσει παλιότερα για να προχωρήσετε στο επόμενο βήμα.

- Τρέξτε το .deb αρχείο για να προσθέσει τα απαραίτητα repositories και meta-data στην source list
- Ενημερώστε την τοπική λίστα με τα repositories, εντολή : `$ sudo apt-get update`



- Εγκαταστήστε το Cuda με την εντολή: `$ sudo apt-get install cuda`
- Αν έχετε εγκατεστημένους τους Nouveau (open source drivers) ίσως χρειαστεί να τους απενεργοποιήσετε . Για να δείτε αν είναι εγκατεστημένοι υπάρχει η εντολή : `$ lsmod | grep nouveau`
- Προσθέστε στο **Bashsource** αρχείο την εντολή που προσθέτει το Cuda σαν μεταβλητή περιβάλλοντος (`$ export PATH=/es/local/cuda-8.0/bin${PATH}:+:${PATH}}`)
- Εν συνεχεία, εγκαταστήστε τα παραδείγματα κώδικα(code samples) με 'άδεια' εγγραφής . Τρέξτε `$ cuda-install-samples-8.0.sh <φάκελος προορισμού>`
- Στο τερματικό , μετακινηθείτε στο φάκελο που έχουν εγκατασταθεί τα samples (`~/NVIDIA_CUDA-8.0_Samples`) και τρέξτε την εντολή `$ make` για να γίνουν compile. Θα δημιουργηθούν τα δυαδικά αρχεία (`~/NVIDIA_CUDA-8.0_Samples/bin`)
- Μεταβείτε στον φάκελο των δυαδικών εκτελέσιμων αρχείων και τρέξτε `./ deviceQuery`

```

nikolas@nikolas-X556UJ: ~/cuda-samples/NVIDIA_CUDA-8.0_Samples/bin/x86_64/linux/release$ ./deviceQuery Starting...
CUDA Device Query (Runtime API) version (CUDART static linking)

Detected 1 CUDA Capable device(s)

Device 0: "GeForce 920M"
  CUDA Driver Version / Runtime Version      8.0 / 8.0
  CUDA Capability Major/Minor version number: 3.5
  Total amount of global memory:             2004 MBytes (2100953088 bytes)
  * ( 2) Multiprocessors, (192) CUDA Cores/MP: 384 CUDA Cores
    GPU Max Clock rate:                     954 MHz (0.95 GHz)
    Memory Clock rate:                      900 Mhz
    Memory Bus Width:                       64-bit
  ★ L2 Cache Size:                         524288 bytes
    Maximum Texture Dimension Size (x,y,z): 1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
    Maximum Layered 1D Texture Size, (num) layers: 1D=(16384), 2048 layers
    Maximum Layered 2D Texture Size, (num) layers: 2D=(16384, 16384), 2048 layers
  ★ Total amount of constant memory:        65536 bytes
  ★ Total amount of shared memory per block: 49152 bytes
    Total number of registers available per block: 65536
  ★ Warp size:                            32
  ★ Maximum number of threads per multiprocessor: 2048
  ★ Maximum number of threads per block:     1024
    Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
    Max dimension size of a grid size (x,y,z): (2147483647, 65535, 65535)
    Maximum memory pitch:                   2147483647 bytes
    Texture alignment:                      512 bytes
    Concurrent copy and kernel execution:   Yes with 1 copy engine(s)
    Run time limit on kernels:              Yes
    Integrated GPU sharing Host Memory:    No
    Support host page-locked memory mapping: Yes
    Alignment requirement for Surfaces:     Yes
    Device has ECC support:                Disabled
    Device supports Unified Addressing (UVA): Yes
    Device PCI Domain ID / Bus ID / location ID: 0 / 1 / 0
    Compute Mode:
      < Default (multiple host threads can use ::cudaSetDevice() with device simultaneously) >

  x DeviceQuery, CUDA Driver = CUDART, CUDA Driver Version = 8.0, CUDA Runtime Version = 8.0, NumDevs = 1, Device0 = GeForce 920M
  result = PASS
nikolas@nikolas-X556UJ: ~/cuda-samples/NVIDIA_CUDA-8.0_Samples/bin/x86_64/linux/release$ 

```

Αν δείτε κάτι τέτοιο στην οθόνη σας η εγκατάσταση έχει ολοκληρωθεί με επιτυχία. Τέλος προτείνετε για την ανάπτυξη κώδικα να χρησιμοποιήσετε το ενσωματωμένο περιβάλλον ανάπτυξης(I.D.E. Integrated Development Environment) nsight. Μπορείτε να το εγκαταστήσετε με την εντολή : `$sudo apt-get install nisight`

| Linux Distribution | Gcc Version | Nvidia drivers | Nvidia gpu | Cuda Version |
|--------------------|-------------|----------------|--------------|--------------|
| LUbuntu 16.04 | 5.4.3 | 367.56 | GeForce 920M | 8.0 |

Σημείωση:

Διευκρινίζεται ότι όταν αναφέρεται η λέξη host εννοούμε την CPU , ενώ όταν αναφέρεται η λέξη device εννοούμε την GPU. Αυτό το κάνουμε γιατί στο παράλληλο προγραμματιστικό μοντέλο ο επεξεργαστής βλέπει την κάρτα γραφικών σαν ένα πυρήνα διαφορετικού τύπου και του παραπέμπει ειδικές συναρτήσεις που λέγονται gru kernels ή αλλιώς device kernels. Επιπλέον επειδή η GPU έχει διαφορετική μνήμη από τον επεξεργαστή δεν μπορεί να χρησιμοποιήσει μεταβλητές που βρίσκονται στην κύρια μήνη του συστήματος. Για λόγους διευκόλυνσης λοιπόν , χρησιμοποιείται συχνά διαφορετική ονομασία για την μεταβλητές της GPU συνήθως με ένα πρόθεμα μπροστά. Για παράδειγμα αν A είναι μια μεταβλητή στην κύρια μνήμη τότε για να την μεταφέρω στην global μνήμη των γραφικών την ονομάζω d_A (d από το device).Έτσι επιτυγχάνεται σαφής διαχωρισμός των δεδομένων που είναι στην κύρια μνήμη και στην μνήμη της κάρτας γραφικών.

Λογισμικό Cuda πυρήνων για νευρωνικά δίκτυα

Link για την πλήρη ανάπτυξη του λογισμικού, στα πλαίσια της διπλωματικής εργασίας, [εδώ](#)

Βασική Υλοποίηση Εμπρόσθιας Διάδοσης

```
_global_ void Kernel_forward( float *A , Matrix W , float *out ) // vector_size = W.width
{
    int tx = threadIdx.x;
    float register sum = 0 ;
    int register k ;
    float register a,b ;

//each thread multiplies one row of W with the input Vector A
#pragma unroll
    for(k=0 ; k<W.width ; k++)
    {
        a = W.elements[ tx*W.width + k ] ;
        b = ( k==(W.width-1) )? 1 : A[k] ;
        sum += ( a * b ) ;
    }
    out[tx]= (nonlin)? Sigmoid(sum) : TanSig(sum) ;
}
```



Εμπρόσθια Διάδοση με γενικευμένη ελάττωση με προσθήκη μηδενικών (zero padding)

```
__global__ void Kernel_forward_fast2( float *A , Matrix W , float *Out )
{
    extern __shared__ float sdata[];
    int bx = blockIdx.x ;
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(W.width)+ threadIdx.x;
    register float temp ;
    ///////////////////////////////////////////////////
    if ( tid<=(W.width-1) )
    {
        temp = (tid==W.width-1)? 1 :A[tid] ; // 1 for bias neuron
        sdata[tid] = (W.elements[i])*temp;
    }
    else
    {
        sdata[tid] = 0 ; // padding in order to be power of 2
    }
    __syncthreads();
    //***** blockDim.x is power of 2 , Not W.width !!!!
    // REDUCE HERE IN SHARED MEMORY- full unroll of for loop - reduction in shared mem
    if (blockDim.x >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
    if (blockDim.x >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
    if (blockDim.x >= 128) { if (tid < 64) { sdata[tid] += sdata[tid + 64]; } __syncthreads(); }

    if (tid < 32)
    {
        if (blockDim.x >= 64) sdata[tid] += sdata[tid + 32];
        if (blockDim.x >= 32) sdata[tid] += sdata[tid + 16];
        if (blockDim.x >= 16) sdata[tid] += sdata[tid + 8];
        if (blockDim.x >= 8) sdata[tid] += sdata[tid + 4];
        if (blockDim.x >= 4) sdata[tid] += sdata[tid + 2];
        if (blockDim.x >= 2) sdata[tid] += sdata[tid + 1];
    }
    // write result for this block to global mem
    if (tid == 0) Out[bx] = (nonlin)? Sigmoid(sdata[0]) : TanSig(sdata[0]) ;
}
```

Εμπρόσθια Διάδοση με γενικευμένη ελάττωση για μεγάλο και άρτιο αριθμό εισόδων

```
__global__ void Kernel_forward_fast3( float *A , Matrix W , float *Out )
{
    extern __shared__ float sdata[];
    int bx = blockIdx.x ;
    unsigned int tid = threadIdx.x;
    unsigned int i = blockIdx.x*(W.width)+ threadIdx.x*2;
    if ( 2*tid<(W.width) )
    {
        float temp1 = ((2*tid)==W.width-1)? 1 :A[tid] ; // 1 for bias neuron
        float temp2 = ((2*tid+1)==W.width-1)? 1 :A[tid+1] ; // 1 for bias neuron
        sdata[tid] = ((W.elements[i])*temp1)+((W.elements[i+1])*temp2); // first reduction add during load
    }
    else
    {
        sdata[tid] = 0 ; // padding in order to be power of 2 for reduction
    }
    __syncthreads();
//***** blockDim.x is power of 2 , Not W.width !!!!
// REDUCE HERE IN SHARED MEMORY- full unroll of for loop - reduction in shared mem
if (blockDim.x >= 512) { if (tid < 256) { sdata[tid] += sdata[tid + 256]; } __syncthreads(); }
if (blockDim.x >= 256) { if (tid < 128) { sdata[tid] += sdata[tid + 128]; } __syncthreads(); }
if (blockDim.x >= 128) { if (tid < 64) { sdata[tid] += sdata[tid + 64]; } __syncthreads(); }
if (tid < 32)
{
    if (blockDim.x >= 64) sdata[tid] += sdata[tid + 32];
    if (blockDim.x >= 32) sdata[tid] += sdata[tid + 16];
    if (blockDim.x >= 16) sdata[tid] += sdata[tid + 8];
    if (blockDim.x >= 8) sdata[tid] += sdata[tid + 4];
    if (blockDim.x >= 4) sdata[tid] += sdata[tid + 2];
    if (blockDim.x >= 2) sdata[tid] += sdata[tid + 1];
}
__syncthreads();
// write result for this block to global mem
if (tid == 0) Out[bx] = Sigmoid(sdata[0]);
}
```

Οπισθοδρομική Διάδοση σφάλματος επιπέδου εξόδου

```

__global__ void Kernel_back_last( float *A , Matrix W , float *Out_ff , float *Desired , float *Delta_val , float lrate )
{
    int tx = threadIdx.x;
    float register DeltaW = 0 ;
    int register k ;
    float error = Out_ff[tx]-Desired[tx];
    float sigmoid_derivative= (nonlin)? Sig_der( Out_ff[tx] ) :TanSig_der( Out_ff[tx] ) ;
    float register delta_neuron = error*sigmoid_derivative ;

    Delta_val[tx] = delta_neuron ;

    float register w_old , w_new, previous_level_output ;
#pragma unroll
    for(k=0 ; k<W.width ; k++)
    {
        w_old = W.elements[ tx*W.width + k ] ;
        previous_level_output = ( ( k==(W.width-1) ) ? 1 : A[k] ) ;
        DeltaW = delta_neuron * previous_level_output * lrate ;
        w_new = w_old - DeltaW ;
        W.elements[ tx*W.width + k ] = w_new;
    }
}

```

Οπισθοδρομική Διάδοση σφάλματος κρυφών επιπέδων



Αναφορές

- Martin T Hagan , & Howard B Demuth. (2014). *Neural Network Design*.
- Braun, M. R. (1994). *A Direct Adaptive Method for Faster Backpropagation Learning*:. Karlsruhe.
- Davis, R. (2017). *Neural Networks and Deep Learning*.
- Dunne, R. A. (2007). *A statistical approach to neural networks for pattern recognition*.
- Hagan, M. T., Demuth, H. B., & Beale, M. H. (2016). *Neural Network Design*.
- Harris, M. (2005). *Optimizing Parallel Reduction in CUDA*. Nvidia.
- Haykin, S. (2009). *Neural Networks & Machine Learning*.
- Heaton, J. (2013). *Artificial Intelligence for Humans, Volume 3, Deep Learning and Neural Networks*.
- Ian Goodfellow, Y. B. (2015). *Deep Learning*.
- Jason Sanders, & Edward Kandrot. (2009). *CUDA by Example An Introduction to GPU programming*.
- Kalman, & Kwasny. (1992). *Why tanh: choosing a sigmoidal function*.
- Kaufmann, M. (2010). *Programming massively parallel processors, a hands-on approach*.
- Kaufmann, M. (2012). *CUDA Programming A Developer's Guide to Parallel Computing with GPUs*.
- Liang, N. B. (1996). *Neural Networks Fundamentals with Graphs, Algorithms and Applications*.
- Nitish Srivastava, & Geoffrey Hinton. (2014). *A Simple Way to Prevent Neural Networks from Overfitting*. Toronto.
- Nvidia. (2016). *CUDA C PROGRAMMING GUIDE Design Guide*.
- Pacheco, P. (2012). *An Introduction to Parallel Programming:Errata* .
- Rashid, T. (2016). *Make Your Own Neural Network*.
- Reeves, C. (1992). *Modern heuristic Techniques for Combinatorial Problems*.
- Rojas, R. (1996). *Fast Learning Algorithms*.
- Samarasinghe, S. (2006). *Neural Networks for Applied Sciences and Engineering: From Fundamentals to Complex Pattern Recognition*.
- Shaffer, S. C. (2015). *Code Your Own Neural Network: A step-by-step explanation*.
- Smart, M. (2017). *Neural Networks for Complete Beginners: Introduction for Neural Network Programming*.
- Wittwer, T. (2006). *An Introduction to Parallel Programming*.
- Δερματάς, Ε. (1997). *Αναγνώριση Προτύπων*.

