Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	ΣΥΓΚΡΙΤΙΚΗ ΜΕΛΕΤΗ ΕΝΑΛΛΑΚΤΙΚΗΣ ΥΛΟΠΟΙΗΣΗΣ ΒΑΘΙΑΣ ΜΑΘΗΣΗΣ ΣΕ ΠΑΡΑΛΛΗΛΑ ΠΕΡΙΒΑΛΛΟΝΤΑ COMPARATIVE STUDY OF ALTERNATIVE IMPLEMENTATION OF DEPTH LEARNING IN PARALLEL ENVIRONMENTS
Ονοματεπώνυμο Φοιτητή	ΜΠΕΑΖΟΓΛΟΥΣ ΙΩΑΝΝΗΣ
Πατρώνυμο	ΝΙΚΟΛΑΟΣ ΜΠΕΑΖΟΓΛΟΥ
Αριθμός Μητρώου	ΜΠΠΛ/14054.
Επιβλέπων	ΤΣΙΧΡΙΝΤΖΗΣ ΓΕΩΡΓΙΟΣ, ΚΑΘΗΓΗΤΗΣ

24 ΟΚΤΩΒΡΙΟΥ 2018

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
ΤΣΙΧΡΙΝΤΖΗΣ ΓΕΩΡΓΙΟΣ ΕΠΙΒΛΕΠΩΝ	ΑΛΕΠΗΣ ΕΥΘΥΜΙΟΣ ΚΑΘΗΓΗΤΗΣ	ΣΩΤΗΡΟΠΟΥΛΟΣ ΔΙΟΝΥΣΗΣ ΚΑΘΗΓΗΤΗΣ

ΠΕΡΙΛΗΨΗ

Η παρούσα εργασία διερευνά συγκριτικά τις διαφορές εναλλακτικής υλοποίησης Βαθιάς Μάθησης αλγορίθμων σε παράλληλα περιβάλλοντα. Βρισκόμαστε στο τρίτο κύμα εξέλιξης Βαθείας Μάθησης με παρόντα τα Συνελικτικά Νευρωνικά Δίκτυα (Convolution Network) τα Big Data (Μεγάλα σε όγκο Δεδομένα) και την ανάπτυξη της τεχνολογίας υλικού με επεξεργαστές μονάδων γραφικών GPU. Λόγω των τελευταίων μπορούμε να ερευνήσουμε τα νευρωνικά δίκτυα Βαθιάς Μάθησης μια και οι βασικές λειτουργίες σε αυτές τις κάρτες γίνονται πολύ γρήγορα σε σχέση με τις κοινές CPU. Εξετάζουμε τα πιο ενεργά frameworks και επιλέγουμε τα καταλληλότερα με βάση την ταχύτητα, αξιοπιστία, φορητότητα, και στο χρόνο υλοποίησης του λογισμικού.

ABSTRACT

This paper explores comparatively the differences in alternative implementation of Deep Learning algorithms in parallel environments. We are in the third wave with Convolution Neural Networks and Big Data presenting and developing hardware technology with GPU graphics processors. Because of the latter, we can investigate the Neural Networks of Deep Learning as the basic functions on these cards are very fast in relation to common CPUs. We review the most active frameworks and choose the most appropriate ones based on speed, reliability, portability, and software implementation time.

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Η Μάθηση είναι θεμελιώδης λειτουργία του ανθρώπινου νου. Ώς τώρα δεν έχουν ανακαλυφθεί όλοι οι μηχανισμοί για τη λειτουργία της. Επιστήμονες του κλάδου της πληροφορικής προσπαθούν να βρουν τρόπους να εκπαιδεύσουν ένα υπολογιστικό σύστημα μια μηχανή να μπορεί να παίρνει αποφάσεις, να ομαδοποιεί, να βρίσκει κανόνες και λύσεις, να εξάγει χαρακτηριστικά από δεδομένα για αρκετούς κλάδους επιστημών όπως η ιατρική, η οικονομία, η φυσική, η ψυχολογία. Για να μπορεί κάποιος ή κάτι να μαθαίνει πρέπει να έχει τη δυνατότητα της επεξεργασίας της πληροφορίας δηλαδή την αντίληψη ,τη δυνατότητα της μνήμης δηλαδή την αποθήκευση της πληροφορίας, και τέλος την σκέψη που περιλαμβάνει όλες τις διαδικασίες σύγκρισης, αιτιότητα, ανάλυση, συμπερασματολογία.

Η Μηχανική Μάθηση έχει ορισθεί με διάφορες έννοιες από ανθρώπους που ανήκουν στον κλάδο της Πληροφορικής. Ο Tom M. Mitchell αναφέρει πως η Μηχανική Μάθηση έχει ως πεδίο την κατασκευή προγραμμάτων, υπολογιστικών συστημάτων, που μπορούν και βελτιώνονται με την εμπειρία. Εδώ και μισό αιώνα περίπου, έχουν δοκιμασθεί αλγόριθμοι Μηχανικής Μάθησης, στην προσπάθεια του ανθρώπου να παράγει Τεχνητή Νοημοσύνη. Ένας κλάδος της Τεχνητής Νοημοσύνης είναι η Μηχανική Μάθηση. Μια από τις πρώτες ανακαλύψεις Μηχανικής Μάθησης είναι το παιχνίδι της ντάμας ή αλλιώς τρίλιζας. Εκεί ένα υπολογιστικό σύστημα μπορεί να υπολογίζει τις πιθανές καταστάσεις του αντιπάλου, καθώς και τις δικές του, ώστε να μπορεί να επιλέγει την καλύτερη κίνηση, με αποτέλεσμα να κερδίσει το παιχνίδι. Η μηχανική μάθηση μπορεί και δουλεύει αρκετά καλά σε προβλήματα που ένας άνθρωπος θα ήταν δύσκολα να λύσει, όχι όμως για όλα τα προβλήματα ακόμη. Σημαντικό θεμέλιο της Μηχανικής Μάθησης είναι η κατασκευή αλγορίθμων όπου και θα δούμε παρακάτω κάποιες από τις τελευταίες εξελίξεις. Η βιομηχανία της πληροφορικής έχει κάνει τεράστια άλματα στην κατασκευή οικονομικότερων, ταχύτερων, μικρότερων σε όγκο, υλικών για επεξεργασία και αποθήκευση σήματος, εικόνας, τραγουδιού, ομιλίας κ.α..

Προγραμματιστικά η Μηχανική Μάθηση έχει αρκετά εργαλεία. τρόπους και μεθόδους και μπορεί να εφαρμόζεται σχεδόν σε όλους τους κλάδους των επιστημών, προσφέροντας μια αυτοματοποιημένη διαδικασία από μία μηχανή. Η Μηχανική Μάθηση σε συγκεκριμένα δεδομένα εξετάζει την απόδοση των αλγορίθμων σχετικά με την συμπεριφορά και το αποτέλεσμα τους, την εξαγωγή χαρακτηριστικών για την αναγνώριση αντικειμένων, την ανάλυση κειμένων, την ομαδοποίηση, την εκτίμηση, να πραγματοποιεί προβλέψεις και άλλες λειτουργίες, όπως να οδηγεί ένα αμάξι, να κρατάει σταθερό το ύψος στο αεροπλάνο. Οι δυνατότητες της είναι ατελείωτες αν ο άνθρωπος καταφέρει να εξηγήσει αλγοριθμικά την σκέψη για κάθε πρόβλημα.

Επίσης ερευνά την κατασκευή αλγορίθμων, από τους οποίους εκπαιδεύεται και μπορεί να κάνει προβλέψεις από δεδομένα είναι η λεγόμενη Μάθηση χωρίς επιτήρηση σε αταξινόμητα δεδομένα.

Ενώ η Βαθιά Μάθηση (Deep Learning) σίγουρα δεν είναι καινούργια, βιώνει τεράστια ανάπτυξη εξαιτίας της σύνθεσης και συνεργασίας μεταξύ νευρωνικών δικτύων και της χρήσης των GPU για να επιταχύνει την εκτέλεσή τους, μιας και χωρίς αυτή δυστυχώς δεν θα μπορούσαν να εκπαιδευτούν σε μικρό χρόνο. Επίσης τα Μεγάλα Δεδομένα (Big Data) συνέβαλαν σε αυτή την ανάπτυξη. Επειδή η Βαθιά Μάθηση βασίζεται σε αλγόριθμους εποπτευόμενης μάθησης και όχι μόνο (εκείνους που εκπαιδεύουν νευρωνικά δίκτυα με παραδείγματα δεδομένων (labeled data), όσο πιο μεγάλος είναι ο όγκος τόσο πιο μεγάλη και η ακρίβεια των αποτελεσμάτων τους.

Η ωρίμανση Λογισμικών υλοποίησης αλγορίθμων (frameworks), δίνει την δυνατότητα με χαμηλό κόστος να εκπαιδεύουμε υπολογιστικά συστήματα για να μπορούν να απαντούν με καλό βαθμό αξιοπιστίας και ακρίβειας. Τα δεδομένα μεγάλης κλίμακας (Big Data), που ξεπερνούν κάποια TFlops, βρίσκονται στην $3^{η}$ γενιά της Μηχανικής Μάθησης (σύμφωνα με τον Yann LeCun). Πρακτικά ένα Deep Learning τεχνητό νευρωνικό δίκτυο έχει περισσότερα κρυφά στρώματα νευρώνων από αυτό της Μηχανικής Μάθησης.

Η βασική λειτουργία των στρωμάτων αυτών είναι η εξαγωγή χαρακτηριστικών διαφόρων επιπέδων. Θα μπορούσε στο πρώτο επίπεδο να είχαμε αναγνώριση ορίων, σε ένα δεύτερο τμήμα του αντικειμένου πιο αφαιρετικά, και όλα αυτά μαζί εκπαιδεύονται στα ενδιάμεσα στρώματα. Αυτά τα στρώματα είναι που εξάγουν χαρακτηριστικά από τα δεδομένα.

Όσο πιο πολλά είναι αυτά τα στρώματα τόσο περισσότερο αυξάνεται το υπολογιστικό κόστος για την εκπαίδευση τους καθώς χρειάζονται αρκετά δεδομένα για να μπορέσουν να είναι αποδοτικά και να μπορούν να απαντούν με ικανοποιητική πιθανότητα. Η εξαγωγή χαρακτηριστικών βοηθά σε μη ταξινομημένα δεδομένα, λύνει τα χέρια, σε προβλήματα που τα δεδομένα θα χρειαζόμασταν αρκετό χρονικό διάστημα να ταξινομηθούν , αν δεν είναι αδύνατον σε ένα λογικό πλαίσιο χρόνου από τον άνθρωπο. Λόγω λοιπόν των παραπάνω έχουν αναπτυχθεί μέθοδοι παράλληλης επεξεργασίας σε επίπεδο υλικού & λογισμικού. Δηλαδή να μπορεί ένα σύστημα να κάνει στην ίδια μονάδα χρόνου παραπάνω από μία πράξη. Σήμερα τα περισσότερα υπολογιστικά συστήματα έχουν 2-8 πυρήνες σε κάθε επεξεργαστή, όμως υπάρχουν και άλλες μητρικές κάρτες που μπορούν να έχουν παραπάνω από έναν. Επίσης με το κατάλληλο λογισμικό μπορούμε να έχουμε ένα δίκτυο (clusters) από υπολογιστικά συστήματα που μπορούν μαζί να λύνουν το ίδιο πρόβλημα, χωρίζοντας τα δεδομένα, να συνεργάζονται και να μοιράζονται την ίδια εργασία, που είναι απαραίτητα για λύση μεγάλων προβλημάτων. Εκτός όμως από την παραλληλοποίηση σε επίπεδο επεξεργαστή ή δικτύου, η πληροφορική χρησιμοποιεί κάρτες γραφικών GPU οι οποίες λόγω της διαφορετικής δομής που έχουν μπορούν και έχουν μερικές εκατοντάδες πυρήνες με μνήμη αρκετά υψηλή που φτάνει τα 32 ή 64 GB ή 128 (giga byte) και μπορούν πολλές κάρτες γραφικών να συνεργαστούν μαζί για την παραγωγή Μηχανικής Μάθησης, ελαχιστοποιούν τον χρόνο της μάθησης τουλάχιστον στο 1/10 και κατ' επέκταση την όποια πρόβλεψη μπορούν να κάνουν.

Οι μονάδες GPU διαφέρουν από τους παραδοσιακούς επεξεργαστές μηχανικά-δομικά. Πρώτον, ένας παραδοσιακός επεξεργαστής μπορεί να περιέχει 4 - 24 CPU γενικής χρήσης,

αλλά μια GPU μπορεί να περιέχει 1.000-4.000 εξειδικευμένους πυρήνες επεξεργασίας δεδομένων. Η υψηλή πυκνότητα των πυρήνων καθιστά την GPU ιδιαίτερα παράλληλη σε σύγκριση με τις παραδοσιακές CPU. Αυτό καθιστά τις GPU ιδανικές για μεγάλα νευρωνικά δίκτυα στα οποία μπορούν να υπολογιστούν ταυτόχρονα πολλοί νευρώνες. Οι μονάδες GPU υπερέχουν επίσης σε λειτουργίες διανυσματικών πλωτών σημείων επειδή οι νευρώνες δεν είναι τίποτα περισσότερο από τον πολλαπλασιασμό και την προσθήκη διανυσμάτων. Όλα αυτά τα χαρακτηριστικά κάνουν νευρωνικά δίκτυα σε GPUs τέλεια παράλληλα, όπου απαιτείται ελάχιστη ή καμία προσπάθεια για να παραλληλισθεί ένας αλγόριθμος Βαθιάς Μάθησης.

Λόγω των παραπάνω η συγκεκριμένη εργασία διερευνά συγκριτικά τις διάφορες εναλλακτικές υλοποίησης των Deep Learning αλγορίθμων σε παράλληλα περιβάλλοντα. Θα δούμε τα frameworks και θα συγκρίνουμε μερικούς από τους τελευταίας γενιάς αλγορίθμων εκμάθησης.

DEEP LEARNING ALGORITHMS

ΕΙΣΑΓΩΓΗ

Η Βαθιά Μάθηση οδηγεί το τρίτο κύμα στην έρευνα Τεχνητής Νοημοσύνης, σύμφωνα με τον Lecun. Αναφέρεται πρόσφατα ότι η Βαθιά Μάθηση ξεπερνά τις πρώιμες επιτυχίες της, στην αναγνώριση προτύπων και προς την κατεύθυνση νέων εφαρμογών σε διάφορους τομείς και βιομηχανίες. Προκειμένου να τεθούν σε εφαρμογή αυτές οι ερευνητικές ιδέες, είναι απαραίτητο το πλαίσιο λογισμικού για Βαθιά Μάθηση. Η εφαρμογή νευρωνικών δικτύων απαιτεί ένα σύνολο εξειδικευμένων δομικών στοιχείων, συμπεριλαμβανομένων των πολυδιάστατων συστοιχιών, λειτουργίες ενεργοποίησης και αυτόνομο υπολογισμό λάθους ή απόκλισης. Για να αποφευχθεί η επανάληψη αυτών των εργαλείων, πολλοί προγραμματιστές χρησιμοποιούν προγράμματα ή frameworks ανοικτού κώδικα όπως θα δούμε παρακάτω. Εργαστήκαμε καθαρά με λογισμικό ανοικτού κώδικα.

Επειδή η Βαθιά Μάθηση χρησιμοποιήθηκε για πρώτη φορά επιτυχώς στους τομείς της ορατότητας στην αναγνώριση ομιλίας, τα υφιστάμενα πλαίσια Βαθιάς Μάθησης σχεδιάστηκαν κυρίως για δίκτυα feed-forward όπως Συνελικτικά Νευρωνικά Δίκτυα (CNNs), τα οποία είναι αποτελεσματικά για την ανάλυση δειγμάτων δεδομένων σταθερού μήκους, όπως για παράδειγμα σε εικόνες.

Πιο πρόσφατα, νέοι τύποι μοντέλων Βαθιάς Μάθησης, εκτός από το Convolutional Neural Networks CNN), έχουν δημιουργηθεί και πολλές άλλες εφαρμογές, που συνεχώς πληθαίνουν, και πρωτοτυπούν, η Μηχανική Μάθηση είναι ένα πολλά υποσχόμενο πεδίο έρευνας.

Επιπλέον, μετά από επαναλαμβανόμενα νευρωνικά δίκτυα Recurrent Neural Networks (RNNs) έδειξαν πολλά υποσχόμενα αποτελέσματα σε δεδομένα, όπως το κείμενο φυσικής γλώσσας.. Τα RNN με μακρά βραχυπρόθεσμη μνήμη (LSTM) χρησιμοποιούνται επί του

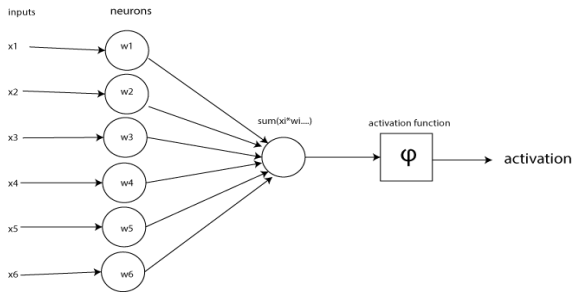
παρόντος με επιτυχία για μετάφραση και μοντέλα συνομιλίας. Αυτό καθιστά δύσκολη την εφαρμογή των νέων μοντέλων.

Όπως για παράδειγμα τα περισσότερα από τα υπάρχοντα frameworks Βαθιάς Μάθησης σχεδιάστηκαν για την επεξεργασία εικόνας χρησιμοποιώντας CNN, τα μοντέλα αυτά της εκπαίδευσης δεν είναι ακόμη βέλτιστα για την εφαρμογή τους.

Η Βαθιά Μάθηση αποτελείται από βαθιά δίκτυα διαφορετικών τοπολογιών. Τα νευρωνικά δίκτυα υπήρξαν εδώ και αρκετό καιρό, αλλά η ανάπτυξη πολυάριθμων επιπέδων δικτύων (κάθε μία από τις οποίες παρέχει κάποια λειτουργία, με βασική την εξαγωγή χαρακτηριστικών) τα έκανε πρακτικότερα για χρήση και πιο αποτελεσματικά. Η προσθήκη στρώσεων σημαίνει περισσότερες διασυνδέσεις και βάρη που σημαίνει περισσότερο υπολογιστικό κόστος,μεταξύ και εντός των στρωμάτων. Αυτό είναι όπου οι GPU ωφελούν τη βαθιά εκμάθηση, καθιστώντας δυνατή την εκπαίδευση και την εκτέλεση αυτών των βαθιών δικτύων (όπου οι επεξεργαστές τύπου CPU δεν είναι τόσο αποδοτικοί, λόγω του ότι μπορούν να υπολογίζουν ταχύτερα τα των πινάκων.

ΤΕΧΝΗΤΟΣ ΝΕΥΡΩΝΑΣ

Πριν μιλήσουμε για του αλγόριθμους θα πρέπει να έχουμε κατανοήσει την λειτουργία ενός τεχνητού νευρώνα. Ένας νευρώνα δέχεται μία έως πολλές εισόδους και έχει 1 έως πολλές εξόδους. Για να έχει έξοδο θα πρέπει να ενεργοποιηθεί, και αυτό γίνεται μόνο όταν λάβει μια μεγαλύτερη τιμή από αυτή που ορίζει η συνάρτηση ενεργοποίησης που βρίσκεται στον πυρήνα του κάθε νευρώνα. Δηλαδή η είσοδος που μπορεί να είναι τα δεδομένα όπως η τιμή φωτεινότητας ενός Pixel μιας εικόνας, μπαίνει ως είσοδο στην συνάρτησης ενεργοποίησης μαζί με άλλα pixel υπολογίζεται και τα αντίστοιχα βάρη τους. Αν ο νευρώνας δεχτεί συνολικά σήμα μεγαλύτερο από το σταθμικό μέσο της συνάρτησης ενεργοποίησης ή μια σταθερά που ορίζουμε ως bias τότε αντιδρά και ενεργοποιείται στέλνοντας σήμα στην επόμενη σύναψη.



ΤΕΧΝΗΤΟΣ ΝΕΥΡΩΝΑ – ΕΙΣΟΔΟΣ & ΣΥΝΑΡΤΗΣΗ ΕΝΕΡΓΟΠΟΙΗΣΗΣ

Πολλοί νευρώνες εκπαιδεύονται στα βάρη που προαναφέραμε και με συνάψεις μεταξύ τους σχηματίζουν ένα νευρωνικό δίκτυο. Φανταστείτε τους νευρώνες σαν μια ομάδα και σε επίπεδα από την εισαγωγή έως το τελικό επίπεδο που έχει ως έξοδο ένας αλγόριθμος.

Μαθηματική αναπαράσταση νευρώνα:

y = g (Θο + Σθμ(XiΘi)) , γ η συνάρτηση ενεργοποίησης, Θο :bias.

ΝΕΥΡΟΝΙΚΑ ΔΙΚΤΥΑ - ΔΟΜΗ

Τα δομικά υλικά της Μηχανικής Μάθησης είναι ο τεχνητός νευρώνας και στον πυρήνα του υπάρχει η συνάρτηση ενεργοποίησης. Στη συνάρτηση ενεργοποίησης προσθέτουμε τα δεδομένα με το αντίστοιχο βάρος, έχουμε δηλαδή ένα διάνυσμα, που αναπροσαρμόζει ο αλγόριθμος ελαχιστοποιώντας την συνάρτηση λάθους. Αυτό γίνεται με την επανάληψη, ο αλγόριθμος προσπαθεί να βρει τα καλύτερα βάρη για κάθε είσοδο με σκοπό την ελαχιστοποίηση του λάθους. Εδώ πρέπει να ορίσουμε το βαθμό Μάθησης (learning rate). Χρησιμοποιούμε το Learning Rate για να αλλάξουμε τις τιμές των βαρών σε περίπτωση που ο αλγόριθμος δεν αποδίδει καλά κατά την περίοδο εκπαίδευσης. Ανανεώνουμε συνεχώς τα βάρη και τα bias έως ότου βρούμε το λιγότερο λάθος. Το επίπεδο μάθησης μας βοηθά να γνωρίζουμε πόσο πολύ έχουμε αλλάξει τα βάρη και τα bias, ουσιαστικά είναι το πόσο πολύ θα αλλάξουμε τα βάρη ή το bias.

ΣΥΝΑΡΤΗΣΕΙΣ ΕΝΕΡΓΟΠΟΙΗΣΗΣ

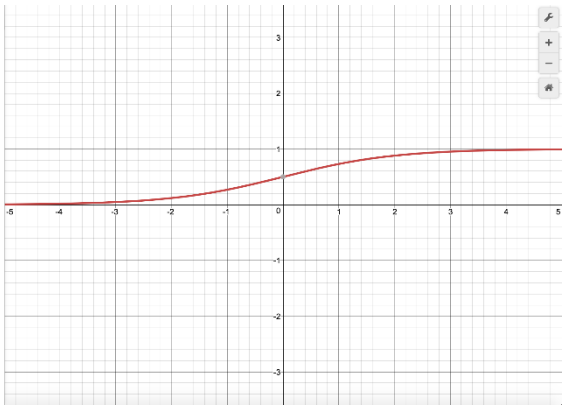
Οι αλγόριθμοι προσπαθούν να μειώσουν το λάθος στο ελάχιστο. Το ελάχιστο μπορούμε πολύ εύκολα να το βρούμε από την παράγωγο της συνάρτησης ενεργοποίησης, εκεί που μηδενίζεται τότε υπάρχει ακρώτατο. Ο αλγόριθμος εκμάθησης θα χρειαστεί n+1 βάρη (ν από τις παρατηρήσεις +1 από το bias)

Η συνάρτηση ενεργοποίησης ουσιαστικά είναι μια συνάρτηση που για οποιεσδήποτε τιμές εισόδου επιστρέφει συνήθως τιμές από 0 έως 1. Τρείς πολύ γνωστές συναρτήσεις που χρησιμοποιούνται στις περισσότερες υλοποιήσεις είναι η σιγμοειδής, η υπερβολική εφαπτομένη και μετά το 2011 είχε προταθεί ως συνάρτηση ενεργοποίησης από τον Ρίτσαρντ Χάνλοσερ, ο ανορθωτής.

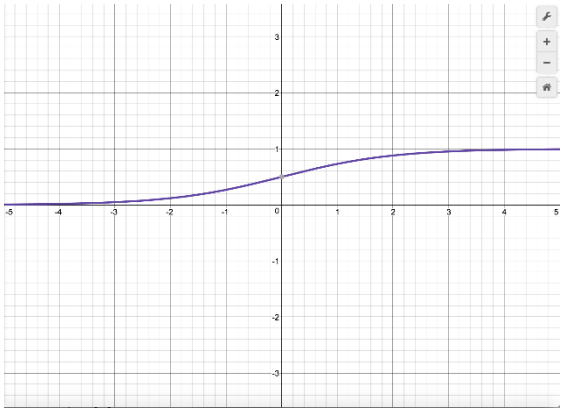
Στη Μηχανική ένας ανορθωτής μετατρέπει το αναλογικό σήμα σε συνεχές και απέδειξε ότι τα δίκτυα Βαθιάς Μάθησης εκπαιδεύονται καλύτερα χρησιμοποιώντας για συνάρτηση ενεργοποίησης παρά την κλασσική σιγμοειδής όπου εμπνεύστηκε από την θεωρία των

πιθανοτήτων. Ο Ανορθωτής ή Rectifier είναι από τους πιο γνωστούς τρόπους ενεργοποίησης των νευρωνικών δικτύων στη Βαθιά Μάθηση.

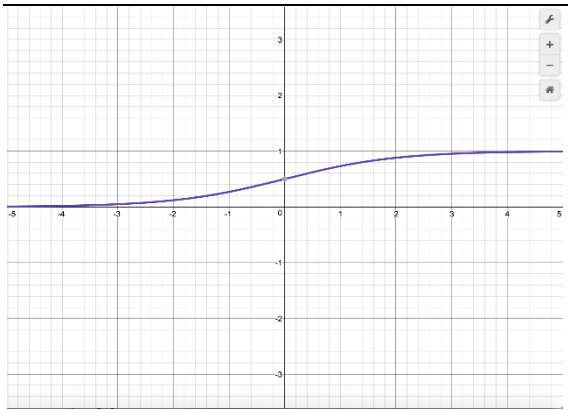
Ένας νευρώνας ονομάζεται Ανορθωμένη Γραμμική Μοναδα (rectified linear unit ή ReLU), και βρίσκει εφαρμογή στην αναγνώριση προτύπων και στην αναγνώριση ομιλίας με χρήση βαθιάς εκμάθησης δικτύων.



Sigmoid



tanh



relu

Ο λόγος που χρησιμοποιούμε αυτές τις συναρτήσεις και όχι τις γραμμικές είναι γιατί οι αλγόριθμοι μπορούν και προσαρμόζονται στα δεδομένα, όταν τα δεδομένα είναι πολλά, και το πρόβλημα σύνθετο αν χρησιμοποιούσαμε γραμμική συνάρτηση ενεργοποίησης τότε η λύση στο πρόβλημα μας να μην ήταν η άριστη. Χρησιμοποιώντας μη γραμμικές συναρτήσεις στους νευρώνες ενός δικτύου τότε μπορούμε να έχουμε μια πιο πολυσύνθετη λύση . Όπως στο παρακάτω παράδειγμα.

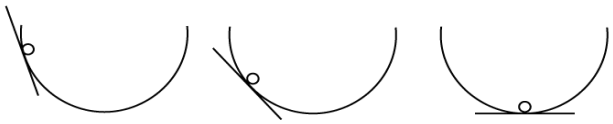
Έχουμε ένα σύνολο από μπλε και κόκκινες κουκκίδες. Ζητάμε από τον υπολογιστή να μας χωρίσει αυτές τις κουκίδες σε 2 ομάδες. Ένα απλό πρόβλημα διασποράς μπορεί να λυθεί πολύ εύκολα με απλή στατιστική και όχι με δίκτυα Βαθιάς Μάθησης, απλώς είναι ένα παράδειγμα για την κατανόηση των τεχνιτών νευρωνικών δικτύων και πώς λειτουργούν. Αν ζητάγαμε από τον υπολογιστή να χωρίσει στη μέση με μια γραμμή αυτά τα δύο σύνολα θα το κάνει με συνάρτηση ενεργοποίησης την y= ax+b, ενώ σε περίπτωση που χρησιμοποιήσουμε μια πιο σύνθετη συνάρτηση θα μπορέσει να δώσει μια λύση πιο ακριβή.

ΣΥΝΑΡΤΗΣΗ ΛΑΘΟΥΣ – ERROR FUNCTION

Στο συγκεκριμένο παράδειγμα αν λυνόταν με δίκτυα Βαθιάς Μάθησης, εκτός από την συνάρτηση ενεργοποίησης, χρειαζόμαστε μια συνάρτηση λάθους. Είναι μια συνάρτηση στην οποία λέμε στον υπολογιστή πόσο καλά τα πήγε. Στόχος της συνάρτησης αυτής είναι να εκπαιδευτούν οι τιμές των βαρών και να βρεθεί μια τιμή της συνάρτησης λάθους πολύ κοντά στο 0. Αυτή θα ήταν η άριστη λύση στο πρόβλημα μας. Στο συγκεκριμένο παράδειγμα ο μέσος όρος της απόστασης όλων των κουκίδων από την γραμμή ορίζει την συνάρτηση λάθους

Error function = 1/nSum(yi – (axi+b)) ^2

Οι συναρτήσεις ενεργοποίησης είναι η παράγωγος της παραπάνω συνάρτησης. Τα ελάχιστα είναι η λύση στο πρόβλημα μας. Συνήθως, αυτή η ελάχιστη τιμή σε πραγματικά προβλήματα είναι αρκετά δύσκολο να βρεθεί σε σύνθετα προβλήματα. Τα τοπικά ελάχιστα προσπαθούν να βρουν οι αλγόριθμοι αναπροσαρμόζοντας σε κάθε κύκλο μάθησης τα βάρη των νευρώνων.



GRADIENTS DESCENT – ERROR FUNCTION

Τα νευρωνικά δίκτυα κατά την υλοποίησή τους ορίζουν τυχαία τα βάρη, αυτά αναπροσαρμόζονται με κάθε back-propagation και αναπροσαρμόζονται, η μεταβολή των βαρών ορίζεται από το βαθμό μάθησης που ορίζουμε προσπαθώντας να βρούμε το ελάχιστο στην συνάρτηση λάθους.

Αρχιτεκτονικές Βαθιάς Μάθησης – Deep Learning

Η αρχιτεκτονική βασίζεται σε διασυνδεδεμένα δίκτυα μεταξύ τους, σύνθετων ή απλών μονάδων. Ο τρόπος σύνδεσης ποικίλει, και πολλές φορές ένας ερευνητής είναι δύσκολο να προσδιορίσει τον αριθμό, ή τα επίπεδα των νευρώνων για εύρεση της καλύτερης λύσης.

Η Βαθιά Μάθηση αντιπροσωπεύεται από ένα φάσμα αρχιτεκτονικών που μπορούν να δημιουργήσουν λύσεις για μία γκάμα προβλημάτων. Βεβαίως το χτίσιμο αυτών των αρχιτεκτονικών είναι αρκετά δύσκολο, υπάρχουν αρκετά εργαλεία ανοιχτού κώδικα στον τομέα της Πληροφορικής και διαμοιρασμένα δωρεάν στο κοινό κάνουν τις υλοποιήσεις αυτές αρκετά ενδιαφέρουσες.

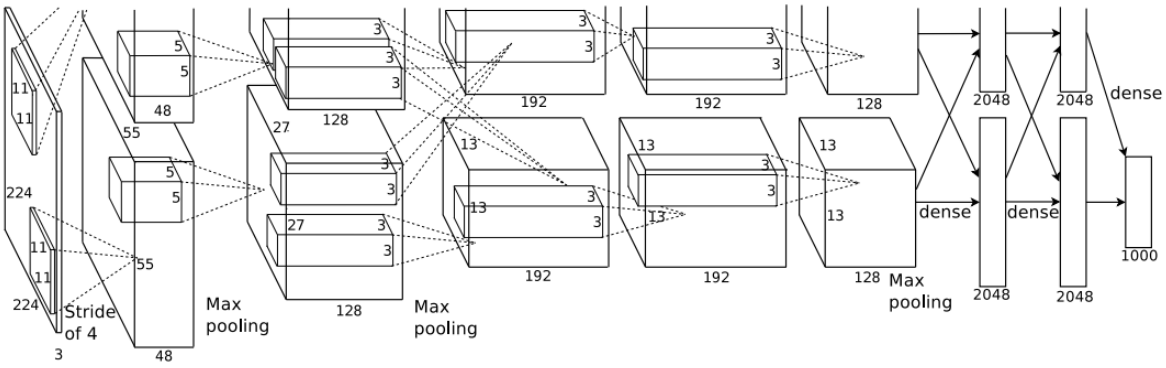
Πολύ γνωστές αρχιτεκτονικές είναι:

Το AlexNet

https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Το AlexNet είναι η πρώτη βαθιά αρχιτεκτονική που εισήγαγε ένας από τους πρωτοπόρους στη Βαθιά Μάθηση – ο Geoffrey Hint. Πρόκειται για μια απλή αλλά ισχυρή αρχιτεκτονική δικτύου, η οποία βοήθησε να προετοιμάσει το δρόμο για πρωτοποριακή έρευνα στην Βαθιά Μάθηση όπως είναι τώρα. Το AlexNet χρησιμοποιήθηκε σε διαγωνισμό το 2010 ώστε να κατηγοριοποιήσει 1.2 εκ. φωτογραφίες σε 1000 κλάσεις.

Η βάση δεδομένων του ImageNet έχει πάνω από 1.2 εκατομύρια φωτογραφίες και 22.000 κατηγορίες το οποίο δεν υπάρχει σε βάσεις δεεδομένων όπως το Cifar 10/100, MNIST ΚΑΙ ΤΟ CALTECH-101.



Alexnet

Καθώς αναλύουμε το AlexNet, το οποίο ανήκει στην κατηγορία CNN (Convolutional Neural Network), μοιάζει με μια απλή αρχιτεκτονική με στρώματα συνελικτικά συγκεντρώνοντας το ένα πάνω στο άλλο, ακολουθούμενο από πλήρως συνδεδεμένα στρώματα στην κορυφή. Η έξοδος του αλγορίθμου είναι ένα softmax regression το οποίο μετατρέπει τα βάρη σε πιθανότητα διανομεί στην κατηγοριοποίηση των φωτογραφιών σε 1000 κλάσεις. Όταν χρησιμοποιούμε το softmax ουσιαστικά κατηγοριοποιούμε τις κλάσεις με αριθμούς, χρησιμοποιώντας μια softmax συνάρτηση ενεργοποίησης. Στο τελευταίο επίπεδο έχουμε ν νευρώνες. Ο κάθε νευρώνας έχει μια πιθανότητα να είναι μια από τις 1000 κλάσεις, αλλιώς λέγεται softmax επίπεδο.

Ως είσοδο, ο αλγόριθμος δέχεται 224χ224χ3 ανάλυση η οποία τροφοδοτεί το πρώτο επίπεδο convolution το οποίο χρησιμοποιεί 96 kernel. Είναι παρόμοιος με το δίκτυο LeNet αλλά με περισσότερα επίπεδα. Εδώ βλέπουμε την πρώτη χρήση της ReLu ως συνάρτησης ενεργοποίησης, με batch size 128 pixel, SGD momentum 0.9, baum;ow ekpa;ideyshw 1e-2 , μεταβλητό ανά 10, L2 weight decay 5e-4. Πρόκειται για μια πολύ απλή αρχιτεκτονική, η οποία διαμορφώθηκε από την δεκαετία του '80. Τα πράγματα που ξεχωρίζουν σ' αυτό το μοντέλο είναι η κλίμακα με την οποία εκτελεί το έργο και τη χρήση GPU για εκπαίδευση. Στη δεκαετία του 1980, η CPU χρησιμοποιήθηκε για την εκπαίδευση ενός νευρωνικού δικτύου. Ενώ στη χρήση του το AlexNet επιταχύνει την εκπαίδευση κατά 10 φορές μόνο με τη χρήση της GPU. Παρόλο που είναι λίγο ξεπερασμένο, το AlexNet εξακολουθεί να χρησιμοποιείται ως σημείο εκκίνησης για την εφαρμογή Βαθιών Νευρωνικών Δικτύων για όλες τις εργασίες, είτε πρόκειται για όραση υπολογιστή είτε για αναγνώριση ομιλίας.

VGG Net

Το δίκτυο VGG εισήχθη από τους ερευνητές του Visual Graphics Group στην Οξφόρδη. Το δίκτυο αυτό χαρακτηρίζεται ιδιαίτερα από ένα σχήμα που μοιάζει με πυραμίδα, όπου τα κάτω στρώματα που είναι πιο κοντά στην εικόνα, είναι ευρείες, ενώ τα ανώτερα στρώματα είναι βαθιά.

VGG NET ARCHITECTURE

Όπως απεικονίζει η εικόνα, το VGG περιέχει μεταγενέστερα συνελικτικά στρώματα ακολουθούμενα από τη συγκέντρωση στρώσεων. Τα στρώματα συγκέντρωσης είναι υπεύθυνα για τη στενότητα των στρωμάτων. Στην εργασία τους, πρότειναν πολλούς τέτοιους τύπους δικτύων, με αλλαγή στην εμβέλεια της αρχιτεκτονικής.

Τα πλεονεκτήματα του VGG είναι:

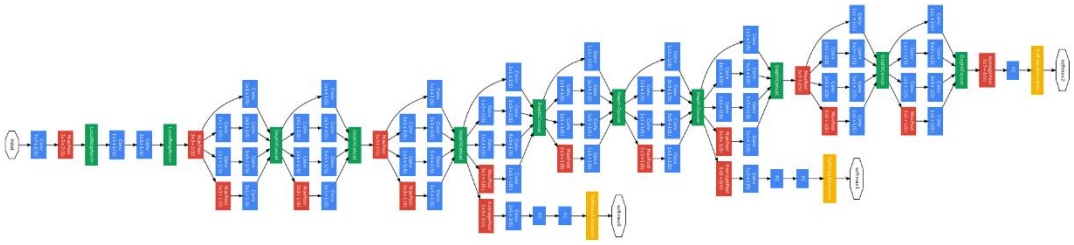
Είναι μια πολύ καλή αρχιτεκτονική για τη συγκριτική αξιολόγηση ενός συγκεκριμένου έργου.

Επίσης, προ-εκπαιδευμένα δίκτυα για VGG είναι διαθέσιμα ελεύθερα στο διαδίκτυο, συνήθως χρησιμοποιούνται υπολογισμένα για διάφορες εφαρμογές.

Από την άλλη πλευρά, το κύριο μειονέκτημα του είναι ότι είναι πολύ αργό να εκπαιδεύεται αν εκπαιδεύεται από το μηδέν. Ακόμα και σε μια αξιοπρεπή GPU, θα χρειαστεί περισσότερο από μία εβδομάδα για να ολοκληρώσουμε την εκμάθηση.

3. GoogleNet

Το GoogleNet ή το Inception Network είναι μια κατηγορία αρχιτεκτονικής που σχεδιάστηκε από ερευνητές της Google. Το GoogleNet ήταν ο νικητής του ImageNet 2014, όπου αποδείχθηκε ένα ισχυρό μοντέλο. Σε αυτή την αρχιτεκτονική, μαζί με το βάθος (περιέχει 22 στρώματα σε σύγκριση με το VGG που είχε 19 στρώματα), οι ερευνητές έκαναν επίσης μια νέα προσέγγιση.



GOOGLE NET

Όπως φαίνεται παραπάνω, είναι μια δραστική αλλαγή από τις διαδοχικές αρχιτεκτονικές που είδαμε προηγουμένως. Σε ένα μόνο στρώμα, υπάρχουν πολλοί τύποι "μηχανών εξαγωγής". Αυτό βοηθά έμμεσα το δίκτυο να αποδώσει καλύτερα, καθώς το δίκτυο στην ίδια την εκπαίδευση έχει πολλές επιλογές για να επιλέξει από την επίλυση της εργασίας. Μπορεί να επιλέξει ή να περιστρέψει την είσοδο ή να την συγκεντρώσει απευθείας.

Η τελική αρχιτεκτονική περιέχει πολλαπλά από αυτά τα αρχικά τμήματα στοιβαζόμενα το ένα πάνω στο άλλο. Ακόμα και η κατάρτιση είναι ελαφρώς διαφορετική στο GoogleNet, καθώς τα περισσότερα από τα κορυφαία στρώματα έχουν το δικό τους στρώμα εξόδου. Αυτή η απόχρωση βοηθά το μοντέλο να συγκλίνει ταχύτερα, καθώς υπάρχει μια κοινή εκπαίδευση καθώς και παράλληλη εκπαίδευση για τα ίδια τα στρώματα. 12 φορές λιγότερες παραμέτρους από το alexnet και λιγότερη δύναμη καθώς έχει καλύτερα αποτελέσματα. Σε σχέση με το ALEXNET μπορεί να είναι μεγαλύτερο σαν μοντέλο όμως χρειάζεται τουλάχιστον την μισή υπολογιστική δύναμη για να μπορεί να υπολογίσει καλύτερα τις εικόνες.. Χρησιμοποιεί πλήρως συνδεδεμένα επίπεδα και κάθε απεικόνιση κρυφού επιπέδου είναι μια απεικόνιση του προηγούμενου επιπέδου.

Τα πλεονεκτήματα του GoogleNet είναι:

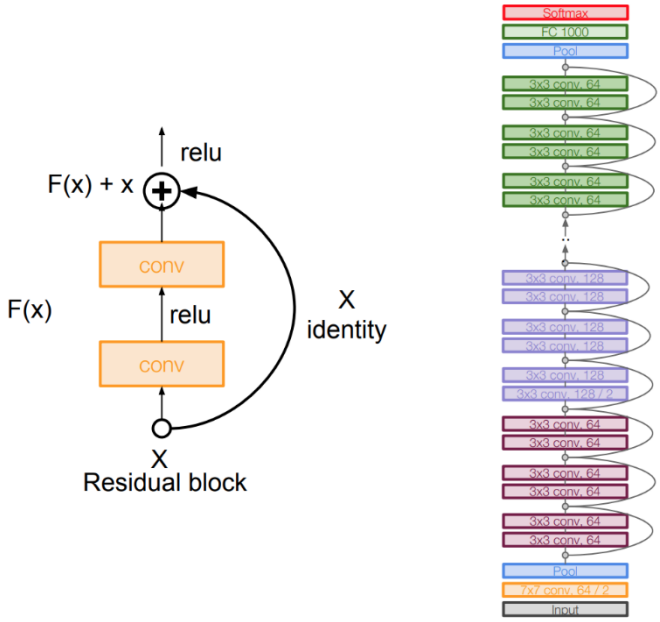
Το GoogleNet εκπαιδεύεται ταχύτερα από το VGG.

Το μέγεθος ενός προ-εκπαιδευμένου GoogleNet είναι συγκριτικά μικρότερο από το VGG. Ένα μοντέλο VGG μπορεί να έχει> 500 MB, ενώ το GoogleNet έχει μέγεθος μόνο 96 MB

Το GoogleNet δεν παρουσιάζει άμεσο μειονέκτημα, αλλά προτείνονται περαιτέρω αλλαγές στην αρχιτεκτονική, οι οποίες καθιστούν το μοντέλο πιο αποτελεσματικό. Μια τέτοια αλλαγή ονομάζεται Δίκτυο Xception, στο οποίο αυξάνεται το όριο της απόκλισης της ενότητας αρχής.

4. ResNet

Το ResNet είναι μια από τις αρχιτεκτονικές που καθορίζουν πραγματικά πόσο βαθιά μπορεί να είναι μια βαθιά αρχιτεκτονική μάθησης. Τα υπολειπόμενα δίκτυα (ResNet εν συντομία) αποτελούνται από πολλαπλές επακόλουθες υπολειπόμενες μονάδες, οι οποίες αποτελούν τη βασική δομική μονάδα της αρχιτεκτονικής ResNet. Μια παράσταση της υπολειπόμενης μονάδας έχει ως εξής:



RESNET

Με απλά λόγια, μια υπολειπόμενη ενότητα έχει δύο επιλογές, είτε μπορεί να εκτελέσει ένα σύνολο λειτουργιών στην είσοδο, είτε μπορεί να παραλείψει αυτό το βήμα εντελώς.

Παρόμοια με το GoogleNet, αυτές οι υπολειπόμενες μονάδες στοιβάζονται η μια πάνω στην άλλη για να σχηματίσουν ένα πλήρες δίκτυο από άκρο σε άκρο.

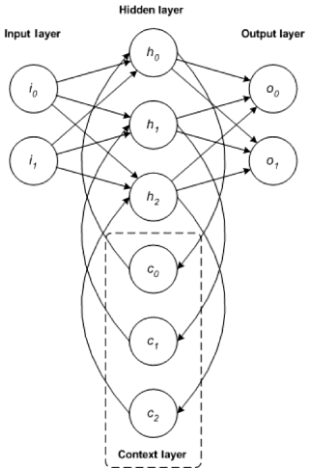
Λίγες ακόμα καινοτόμες τεχνικές που εισήχθησαν από το ResNet είναι:

Χρήση τυπικών SGD αντί για μια εξειδικευμένη τεχνική προσαρμοστικής μάθησης. Αυτό γίνεται μαζί με μια λογική λειτουργία αρχικοποίησης που διατηρεί την εκπαίδευση ανέπαφη. Το κύριο πλεονέκτημα του ResNet είναι ότι εκατοντάδες, ακόμη και χιλιάδες από αυτά τα υπόλοιπα επίπεδα μπορούν να χρησιμοποιηθούν για τη δημιουργία ενός δικτύου και στη συνέχεια να εκπαιδεύονται. Αυτό είναι λίγο διαφορετικό από τα συνηθισμένα διαδοχικά δίκτυα, όπου υπάρχουν μειωμένες αναβαθμίσεις απόδοσης καθώς αυξάνεται ο αριθμός των επιπέδων.

6. RCNN (Region Based CNN)

https://arxiv.org/pdf/1506.01497.pdf

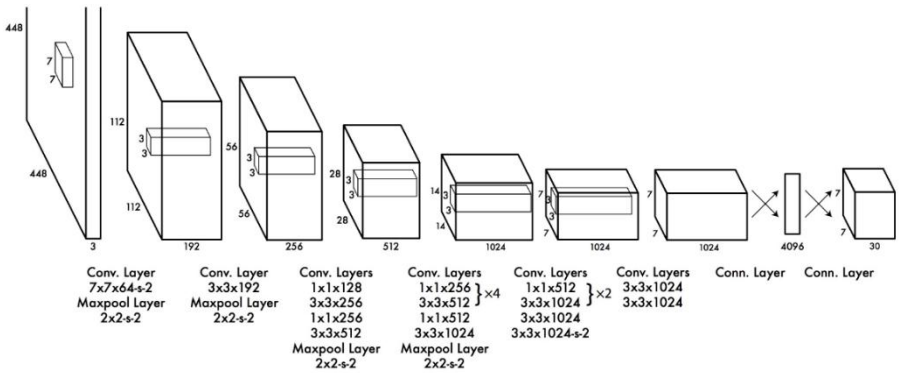
Η βασισμένη στην αρχιτεκτονική του CNN λέγεται ότι είναι η πιο σημαντική από όλες τις αρχιτεκτονικές Βαθιάς Μάθησης που έχουν εφαρμοστεί στο πρόβλημα ανίχνευσης αντικειμένων. Για να λυθεί το πρόβλημα ανίχνευσης, αυτό που κάνει το RCNN είναι να προσπαθήσει να σχεδιάσει ένα πλαίσιο οριοθέτησης πάνω από όλα τα αντικείμενα που υπάρχουν στην εικόνα και στη συνέχεια να αναγνωρίσει ποιο αντικείμενο βρίσκεται στην εικόνα. Λειτουργεί ως εξής:



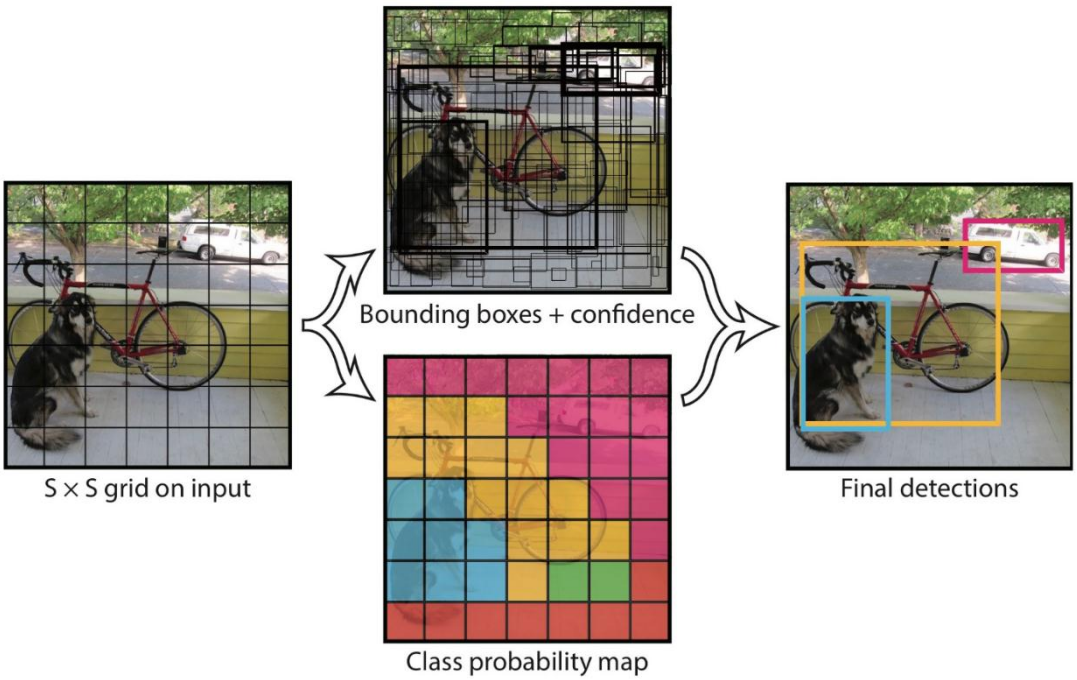
RCNN (Region Based CNN)

YOLO (βλέπετε μόνο μία φορά)

Το YOLO είναι το σημερινό σύστημα πραγματικού χρόνου που βασίζεται σε προηγμένες τεχνολογίες, βασισμένο στη Βαθιά Μάθηση για την επίλυση προβλημάτων ανίχνευσης εικόνων. Όπως φαίνεται στην παρακάτω εικόνα, διαιρεί πρώτα την εικόνα σε καθορισμένα πλαίσια οριοθέτησης και στη συνέχεια εκτελεί παράλληλα έναν αλγόριθμο αναγνώρισης για όλα αυτά τα πλαίσια για να προσδιορίσει σε ποια κατηγορία αντικειμένων ανήκουν. Μετά την αναγνώριση αυτών των τάξεων, προχωρά στη συγχώνευση αυτών των πλαισίων με έξυπνο τρόπο για να διαμορφώσει ένα βέλτιστο πλαίσιο οριοθέτησης γύρω από τα αντικείμενα.



YOLO



YOLO

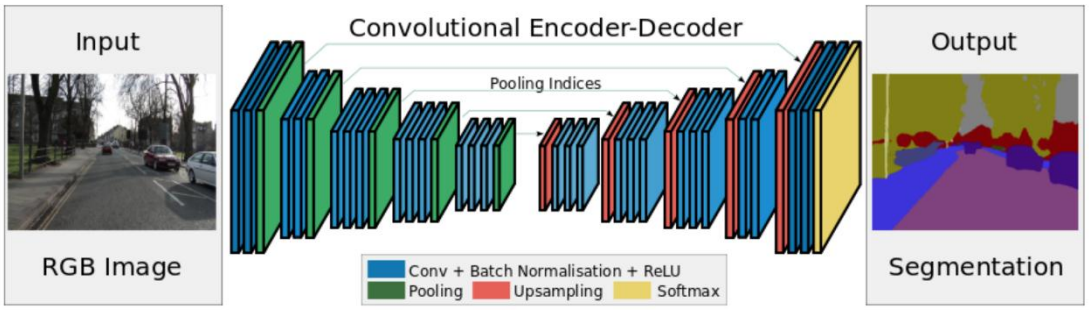
YOLO

Όλα αυτά γίνονται παράλληλα, έτσι ώστε να μπορούν να τρέξουν σε πραγματικό χρόνο. επεξεργασία μέχρι 40 εικόνες ανά δευτερόλεπτο.

Παρόλο που παρέχει μειωμένη απόδοση από το αντίστοιχο RCNN, εξακολουθεί να έχει το πλεονέκτημα ότι είναι πραγματικό χρόνο να είναι βιώσιμο για χρήση σε καθημερινά προβλήματα. Ακολουθεί μια αναπαράσταση της αρχιτεκτονικής του.

SegNet

Το SegNet είναι μια αρχιτεκτονική Βαθιάς Μάθησης που εφαρμόζεται για την επίλυση προβλημάτων τμηματοποίησης εικόνας. Αποτελείται από ακολουθία στρώσεων επεξεργασίας (κωδικοποιητές) ακολουθούμενες από αντίστοιχο σύνολο αποκωδικοποιητών για ταξινόμηση σε pixelwise. Κάτω από την εικόνα συνοψίζεται η λειτουργία του SegNet.



SEGNET

Ένα βασικό χαρακτηριστικό του SegNet είναι ότι διατηρεί λεπτομέρειες υψηλής συχνότητας σε κατακερματισμένη εικόνα, καθώς οι δείκτες συγκέντρωσης του δικτύου κωδικοποιητών συνδέονται με την συγκέντρωση δεικτών δικτύων αποκωδικοποίησης. Εν ολίγοις, η μεταφορά πληροφοριών είναι άμεση, αντί να τις περιστρέφει. Το SegNet είναι το καλύτερο μοντέλο που χρησιμοποιείται όταν αντιμετωπίζουμε προβλήματα κατάτμησης εικόνας

Recurrent Neural Networks (RNNs)

Το RNN είναι μια από τις βασικές αρχιτεκτονικές δικτύου από τις οποίες χτίζονται άλλες αρχιτεκτονικές Βαθιάς Μάθησης. Η κύρια διαφορά μεταξύ ενός τυπικού δικτύου πολλαπλών στρώσεων και ενός επαναλαμβανόμενου δικτύου είναι ότι μάλλον παρά τις πλήρεις συνδέσεις προώθησης προς τα εμπρός, ένα επαναλαμβανόμενο δίκτυο μπορεί να έχει συνδέσεις που τροφοδοτούν εκ νέου σε προηγούμενα στρώματα (ή στο ίδιο στρώμα). Αυτή η ανατροφοδότηση επιτρέπει στους RNNs να διατηρούν τη μνήμη των παρελθουσών εισόδων.

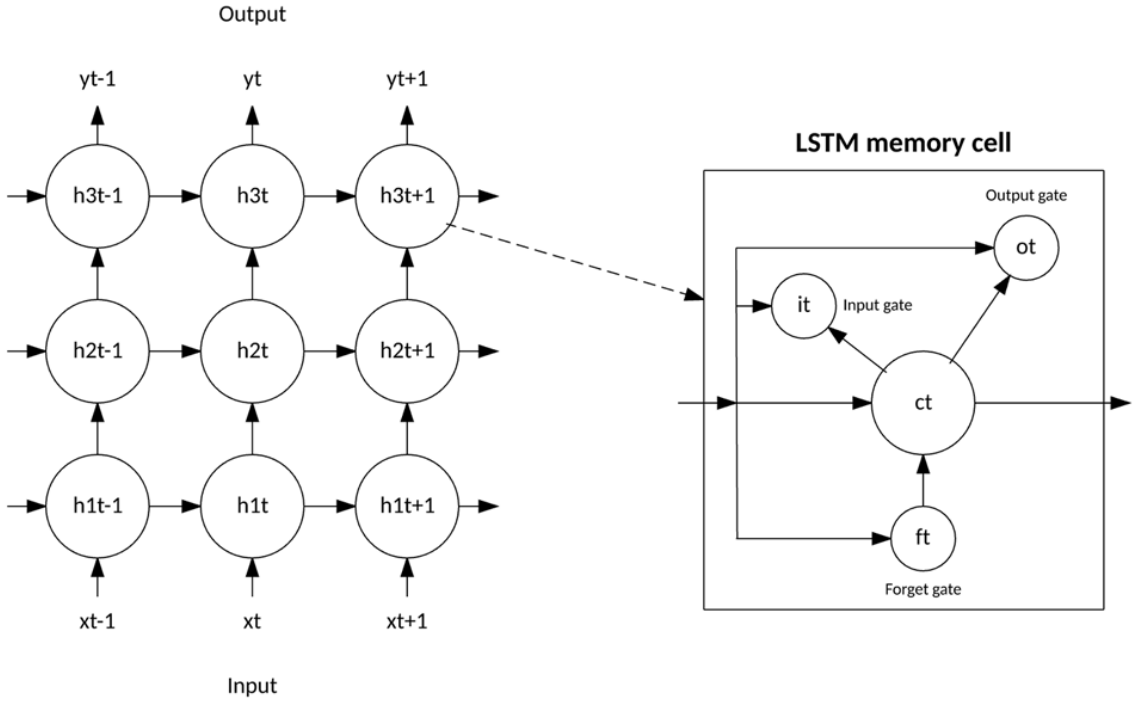
Το RNN αποτελείται από ένα πλούσιο δίκτυο συνόλου αρχιτεκτονικών θα δούμε μια δημοφιλή τοπολογία που ονομάζεται LSTM. Η βασική διαφορά είναι η ανατροφοδότηση εντός του δικτύου, η οποία θα μπορούσε να εκδηλωθεί από ένα κρυφό στρώμα, το στρώμα εξόδου ή κάποιο συνδυασμό αυτών. Τα RNN μπορούν να ξεδιπλωθούν εγκαίρως και να εκπαιδεύονται με τυποποιημένη οπίσθια διάδοση ή χρησιμοποιώντας μια παραλλαγή της οπίσθιας διάδοσης που ονομάζεται BPTT (back-propagation in time).

Long Short-Term Memory (LSTM)/gated recurrent unit (GRU)

Το LSTM δημιουργήθηκε το 1997 από τους Hochreiter και Schimdhuber, αλλά τα τελευταία χρόνια έχει εξελιχθεί σε δημοτικότητα ως αρχιτεκτονική RNN για διάφορες εφαρμογές..

Το LSTM βασίζεται σε νευρώνες και εισήγαγε την ιδέα ενός κυττάρου μνήμης. Η κυψέλη μνήμης μπορεί να διατηρήσει την τιμή της για μικρό ή μεγάλο χρονικό διάστημα ως συνάρτηση

των εισόδων της, πράγμα που επιτρέπει στο κύτταρο να θυμάται τι είναι σημαντικό και όχι μόνο την τελευταία υπολογισμένη αξία του.



LSTM

ΚΥΤΤΑΡΟ ΜΝΗΜΗΣ LSTM

Το κύτταρο μνήμης LSTM περιέχει τρεις πύλες που ελέγχουν τον τρόπο ροής πληροφοριών μέσα ή έξω από το κελί. Η πύλη εισόδου ελέγχει αν μπορούν να εισέλθουν νέες πληροφορίες στη μνήμη. Η θυρίδα ελέγχου όταν ξεχαστεί μια υπάρχουσα πληροφορία, επιτρέπει στο κύτταρο να θυμάται τα νέα δεδομένα. Τέλος, η πύλη εξόδου ελέγχει όταν οι πληροφορίες που περιέχονται στο κελί χρησιμοποιούνται στην έξοδο από το κελί. Το κελί περιέχει επίσης βάρη, τα οποία ελέγχουν κάθε πύλη. Ο αλγόριθμος εκπαίδευσης, συνήθως BPTT, βελτιστοποιεί αυτά τα βάρη με βάση το προκύπτον σφάλμα εξόδου δικτύου.

Το 2014, εισήχθη μια απλοποίηση του LSTM που ονομάζεται περιοδική επαναλαμβανόμενη μονάδα. Αυτό το μοντέλο έχει δύο πύλες, ξεφορτώνοντας την πύλη εξόδου που υπάρχει στο μοντέλο LSTM. Για πολλές εφαρμογές, η GRU(Gate recurrent Unit) έχει απόδοση παρόμοια με την LSTM, αλλά είναι απλούστερη με λιγότερα βάρη και ταχύτερη εκτέλεση.

Η GRU περιλαμβάνει δύο πύλες: μια πύλη ενημέρωσης και μια πύλη επαναφοράς. Η πύλη ενημέρωσης υποδεικνύει πόσα από τα προηγούμενα περιεχόμενα κυψέλης πρέπει να διατηρούνται. Η πύλη επαναφοράς καθορίζει τον τρόπο ενσωμάτωσης της νέας εισόδου με τα

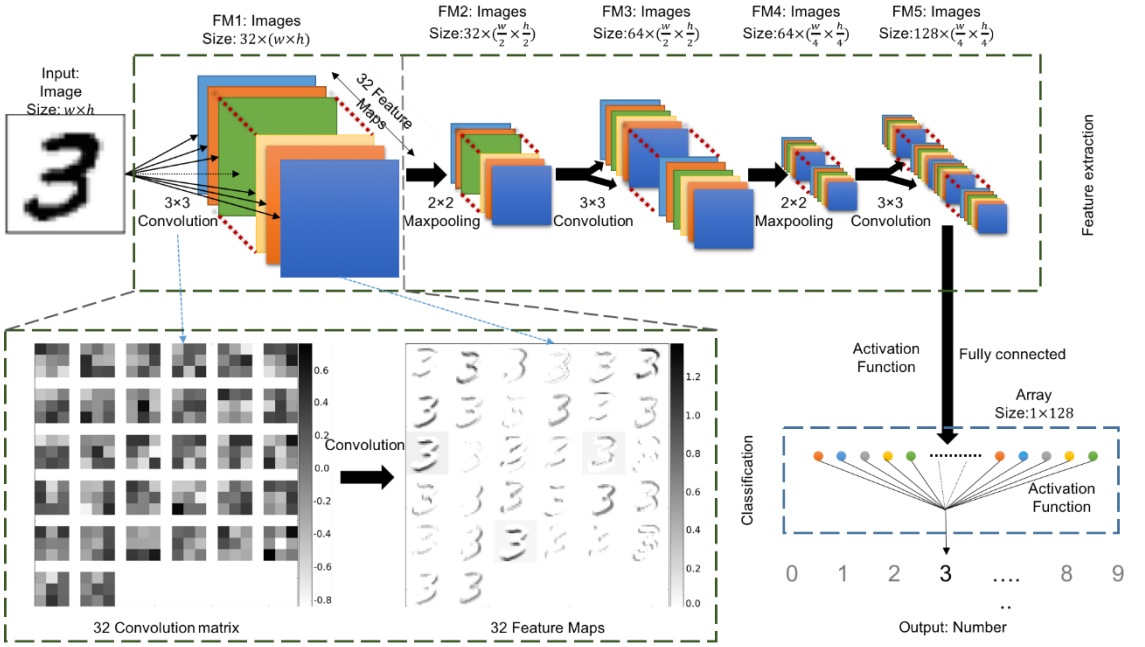
προηγούμενα περιεχόμενα κυψέλης.
Η GRU είναι απλούστερη από το LSTM, μπορεί να εκπαιδευτεί πιο γρήγορα και μπορεί να είναι πιο αποτελεσματική στην εκτέλεση του. Ωστόσο, το LSTM μπορεί να είναι πιο εκφραστικό και με περισσότερα δεδομένα, μπορεί να οδηγήσει σε καλύτερα αποτελέσματα.

Convolutional Neural Networks (CNNs) - Συνελικτικά Νευρωνικά Δίκτυα

Το CNN είναι ένα νευρωνικό δίκτυο με πολλά στρώματα το οποίο ήταν βιολογικά εμπνευσμένο από τον ζωτικό οπτικό φλοιό. Η αρχιτεκτονική είναι ιδιαίτερα χρήσιμη στις εφαρμογές επεξεργασίας εικόνας. Το πρώτο CNN δημιουργήθηκε από τον Yann LeCun. κατά την εποχή εκείνη, η αρχιτεκτονική επικεντρώθηκε στην χειρόγραφη αναγνώριση χαρακτήρων, όπως η διερμηνεία του ταχυδρομικού κώδικα.

Ως ένα βαθύ δίκτυο, τα πρώιμα στρώματα αναγνωρίζουν λειτουργίες (όπως οι άκρες) και τα μεταγενέστερα στρώματα ανασυνθέτουν αυτά τα χαρακτηριστικά σε ιδιότητες υψηλότερου επιπέδου της εισόδου.

Η αρχιτεκτονική LeNet CNN αποτελείται από διάφορα στρώματα που υλοποιούν την εξαγωγή χαρακτηριστικών και στη συνέχεια ταξινομούνται (δείτε την παρακάτω εικόνα). Η εικόνα χωρίζεται σε πεδία ευαισθησίας που τροφοδοτούν σε ένα στρώμα περιελίξεων, το οποίο στη συνέχεια εξάγει χαρακτηριστικά από την εικόνα εισόδου. Το επόμενο βήμα είναι η συγκέντρωση, η οποία μειώνει τη διάσταση των εξαγόμενων χαρακτηριστικών (μέσω δειγματοληψίας) διατηρώντας ταυτόχρονα τις πιο σημαντικές πληροφορίες (συνήθως μέσω της μέγιστης συγκέντρωσης). Ακολούθως πραγματοποιείται ένα άλλο βήμα συνέλιξης και συγκέντρωσης που τροφοδοτεί ένα πλήρως συνδεδεμένο δίκτυο πολλαπλών στρώσεων. Το τελικό επίπεδο εξόδου αυτού του δικτύου είναι ένα σύνολο κόμβων που αναγνωρίζουν τα χαρακτηριστικά της εικόνας (στην περίπτωση αυτή, ένας κόμβος ανά αναγνωρισμένο αριθμό). Εκπαιδεύεται το δίκτυο με τη χρήση οπίσθιου πολλαπλασιασμού.



CNN – CONVOLUTION

Η χρήση βαθιών στρωμάτων επεξεργασίας, συρραφής, συγκέντρωσης και ένα πλήρως συνδεδεμένο στρώμα ταξινόμησης άνοιξε την πόρτα σε διάφορες νέες εφαρμογές νευρωνικών δικτύων Βαθιάς Μάθησης. Εκτός από την επεξεργασία εικόνων, το CNN εφαρμόστηκε επιτυχώς στην αναγνώριση βίντεο και σε διάφορες εργασίες επεξεργασίας φυσικής γλώσσας.

Οι πρόσφατες εφαρμογές των CNNs και LSTMs παρήγαγαν συστήματα υποτίτλων εικόνας και βίντεο στα οποία μια εικόνα ή βίντεο συνοψίζονται στη φυσική γλώσσα. Το CNN υλοποιεί την επεξεργασία εικόνας ή βίντεο και το LSTM εκπαιδεύεται για να μετατρέψει την έξοδο του CNN σε φυσική γλώσσα.

Deep Belief Networks (DBN) - Δίκτυα Βαθιάς Πίστης

Το DBN είναι μια τυπική αρχιτεκτονική δικτύου, αλλά περιλαμβάνει έναν νέο αλγόριθμο εκπαίδευσης. Το DBN είναι ένα δίκτυο πολλαπλών στρώσεων (συνήθως βαθιά, συμπεριλαμβανομένων πολλών κρυφών επιπέδων) στα οποία κάθε ζεύγος συνδεδεμένων στρώσεων είναι μια περιορισμένη μηχανή Boltzmann (RBM).

Με αυτό τον τρόπο, το DBN αντιπροσωπεύεται ως στοίβα των RBM.

RBM (Restricted Boltzmann Machines), η μηχανή Boltzmann έχει συγκεκριμένη μορφή λογαριθμικού γραμμικού μοντέλου Markov Random Field, και είναι παρόμοιο με αυτό του Μπέιζ, Bayes. Οι διαφορές είναι ότι στο Bayesian δίκτυο οι συνάψεις είναι άμεσες και ακυκλικές, ενώ στα δίκτυα Markov δεν έχουν κατεύθυνση και μπορεί να είναι κυκλικές.

Στο DBN, το στρώμα εισόδου αντιπροσωπεύει τις πρώτες αισθητικές εισόδους και κάθε κρυμμένο στρώμα μαθαίνει αφηρημένες παραστάσεις αυτής της εισόδου. Το στρώμα εξόδου, το οποίο αντιμετωπίζεται κάπως διαφορετικά από τα άλλα στρώματα, εφαρμόζει την ταξινόμηση δικτύου. Η εκπαίδευση πραγματοποιείται σε δύο στάδια: προετοιμασία χωρίς επίβλεψη και εποπτεία της τελειοποίησης.

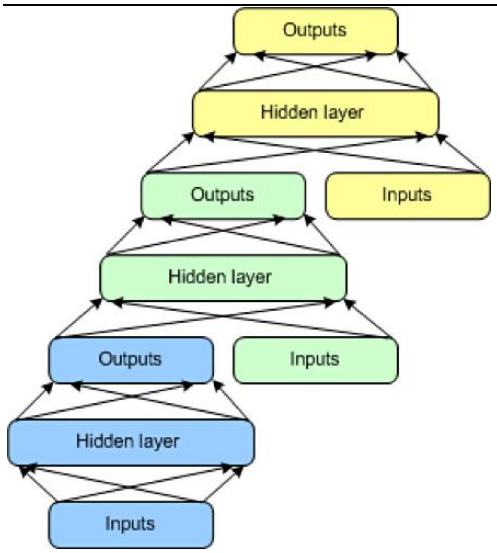
Οι πρόσφατες εφαρμογές των CNNs και LSTMs παρήγαγαν συστήματα υποτίτλων εικόνας και βίντεο στα οποία μια εικόνα ή βίντεο συνοψίζονται στη φυσική γλώσσα. Το CNN υλοποιεί την επεξεργασία εικόνας ή βίντεο και το LSTM εκπαιδεύεται για να μετατρέψει την έξοδο του CNN σε φυσική γλώσσα.

Σε μη προληπτική προετοιμασία, κάθε RBM εκπαιδεύεται για να ανακατασκευάσει την είσοδό του (για παράδειγμα, ο πρώτος RBM ανασυνθέτει το στρώμα εισόδου στο πρώτο κρυφό στρώμα). Ο επόμενος μηχανισμός RBM εκπαιδεύεται παρομοίως, αλλά το πρώτο κρυφό στρώμα αντιμετωπίζεται ως το επίπεδο εισόδου (ή ορατό) και ο RBM εκπαιδεύεται χρησιμοποιώντας τις εξόδους του πρώτου κρυμμένου στρώματος ως εισροές. Αυτή η διαδικασία συνεχίζεται έως ότου προωθηθεί κάθε στρώμα. Όταν ολοκληρωθεί η προ-κατάρτιση, αρχίζει η τελειοποίηση. Σε αυτή τη φάση, οι κόμβοι εξόδου εφαρμόζουν ετικέτες για να τους δώσουν νόημα (αυτό που αντιπροσωπεύουν στο πλαίσιο του δικτύου). Η πλήρης κατάρτιση δικτύου εφαρμόζεται έπειτα χρησιμοποιώντας είτε την εκμάθηση κλίσης είτε την οπίσθια διάδοση για να ολοκληρωθεί η διαδικασία κατάρτισης.

Deep Stacking Networks (DSNs) - Δίκτυα Βαθιάς Στοίβαξης

Η τελική αρχιτεκτονική είναι το DSN, που ονομάζεται επίσης και Βαθύ Κυρτό Δίκτυο. Ένα DSN είναι διαφορετικό από τα παραδοσιακά πλαίσια Βαθιάς Μάθησης στο ότι, αν και αποτελείται από ένα βαθύ δίκτυο, είναι στην πραγματικότητα ένα βαθύ σύνολο επιμέρους δικτύων, το καθένα με τα δικά του κρυμμένα στρώματα. Αυτή η αρχιτεκτονική αποτελεί απάντηση σε ένα από τα προβλήματα της πολυπλοκότητας εκπαίδευσης της Βαθιάς Μάθησης. Κάθε στρώμα σε μια βαθιά αρχιτεκτονική μάθησης αυξάνει εκθετικά την πολυπλοκότητα της εκπαίδευσης, οπότε το DSN θεωρεί την κατάρτιση όχι ως ένα ενιαίο πρόβλημα αλλά ως ένα σύνολο ατομικών προβλημάτων κατάρτισης.

Το DSN αποτελείται από ένα σύνολο ενοτήτων, το καθένα από τα οποία είναι ένα υποδίκτυο στην ολική ιεραρχία του DSN. Σε μια περίπτωση αυτής της αρχιτεκτονικής, δημιουργούνται τρεις μονάδες για το DSN. Κάθε ενότητα αποτελείται από ένα στρώμα εισόδου, ένα μόνο κρυφό στρώμα και ένα στρώμα εξόδου. Οι ενότητες στοιβάζονται το ένα πάνω στο άλλο, όπου οι είσοδοι μιας μονάδας αποτελούνται από τις προηγούμενες εξόδους του στρώματος και το αρχικό διάνυσμα εισόδου. Αυτή η διαστρωμάτωση επιτρέπει στο συνολικό δίκτυο να μάθει πιο πολύπλοκη ταξινόμηση από ό, τι θα ήταν εφικτό δεδομένης μίας μονάδας.



DEEP STACKING NETWORKS

Το DSN επιτρέπει την ατομική κατάρτιση των επιμέρους ενοτήτων, καθιστώντας την αποδοτική δεδομένης της δυνατότητας να εκπαιδεύεται παράλληλα. Η εποπτευόμενη εκπαίδευση εφαρμόζεται ως αναπαράσταση για κάθε ενότητα και όχι ως οπίσθια διάδοση σε ολόκληρο το δίκτυο. Για πολλά προβλήματα, τα DSN μπορούν να αποδίδουν καλύτερα από τα τυπικά DBN, καθιστώντας τα μια δημοφιλής και αποδοτική αρχιτεκτονική δικτύου.

ΣΥΜΠΕΡΑΣΜΑ

Πολλές λύσεις Βαθιάς Μάθησης - ιδιαίτερα σύγχρονης τεχνολογίας - συνδυάζουν πολλαπλά μοντέλα διαφορετικών τύπων αρχιτεκτονικής. η έξοδος ενός μοντέλου μεταβιβάζεται σε άλλο μοντέλο.

Για παράδειγμα, μπορείτε να περάσετε τα δεδομένα εισόδου σε ένα μοντέλο CNN πριν από ένα μοντέλο RNN και στη συνέχεια να περάσετε τελικά αυτή την έξοδο σε ένα MLP που περιλαμβάνει το στρώμα εξόδου.

FRAMEWORKS

1. Caffe – Caffe2

Το Caffe είναι ένα βαθύ πλαίσιο μάθησης ανοιχτού κώδικα που υποστηρίζεται από διεπαφές όπως C, C ++, Python και MATLAB καθώς και τη διεπαφή γραμμής εντολών. Είναι γνωστό για την ταχύτητα και τη μεταφορά του και τη δυνατότητα εφαρμογής του στη μοντελοποίηση νευρωνικών δικτύων συνέλιξης (CNN). Το μεγαλύτερο πλεονέκτημα της χρήσης της βιβλιοθήκης C ++ του Caffe (που συνοδεύεται από Python interface) είναι η δυνατότητα πρόσβασης σε διαθέσιμα δίκτυα από το βαθύ καθαρό αποθετήριο Caffe Model Zoo που είναι προ-εκπαιδευμένο και μπορεί να χρησιμοποιηθεί αμέσως. Όταν πρόκειται για τη μοντελοποίηση CNN ή την επίλυση προβλημάτων επεξεργασίας εικόνας, αυτή θα πρέπει να είναι η βιβλιοθήκη που πρέπει να επιλεχθεί.

Το μεγαλύτερο πλεονέκτημα του Caffe είναι η ταχύτητα. Μπορεί να επεξεργάζεται πάνω από 60 εκατομμύρια εικόνες ημερησίως με μία GPU Nvidia K40. Αυτό είναι 1 ms / εικόνα για εξαγωγή συμπερασμάτων και 4 ms / εικόνα για εκμάθηση - και πιο πρόσφατες εκδόσεις βιβλιοθήκης είναι ακόμα πιο γρήγορα.

Το Caffe είναι ένα δημοφιλές δίκτυο βαθιάς εκμάθησης για οπτική αναγνώριση. Ωστόσο, δεν υποστηρίζει κάποια δίκτυα όπως αυτά που βρίσκονται στο TensorFlow ή το CNTK. Δεδομένης της αρχιτεκτονικής, της συνολικής υποστήριξης για επαναλαμβανόμενα δίκτυα και της γλωσσικής μοντελοποίησης, είναι αρκετά φτωχή και ο καθορισμός πολύπλοκων τύπων στρώσεων πρέπει να γίνει σε γλώσσα χαμηλού επιπέδου.

2. TensorFlow

Το TensorFlow είναι πολύ δυνατό περιβάλλον Μηχανικής Μάθησης με πλούσια παραδείγματα πολύ γρήγορες υλοποιήσεις και είναι γραμμένο σε C++, με χρήση GPU cuda της NVIDIA. Αφού εγκατασταθεί στο λειτουργικό μας σύστημα μπορεί να χρησιμοποιηθεί μέσω της Python. Υποστηρίζει λειτουργικά συστήματα Linux , Windows και Macindos χωρίς GPU ακόμα και Rasbian. Επίσης, είναι έτοιμο προς υλοποίηση κάθε βιβλιοθήκης μέσω του Docker. Το Docker είναι ένα είδωλο που περιέχει όλα τα απαραίτητα για να μπορεί να εκτελεστεί τοπικά. Υπάρχει μεγάλη ενεργητικότητα και υιοθετεί τις τελευταίες εξελίξεις, υποστηρίζει αρκετούς από τους παραπάνω αλγόριθμους που αναφέραμε, και μπορεί να εγκατασταθεί ευκολότερα σε λειτουργικό Linux Ubuntu 16.01.

3. APACHE SINGA

Το Singa μπορεί είναι και αυτό ένα ανοικτού κώδικα περιβάλλοντος εργασίας Μηχανικής Μάθησης, υποστηρίζει και αυτό μάθηση με χρήση GPU, έχει πολύ καλό documentation, και τρέχει CNN χρησιμοποιώντας πολλές βάσεις δεδομένων. Μπορεί να συνδεθεί και να χρησιμοποιηθεί κατ' ευθείαν μέσω της γλώσσας python και από τα πιο δυνατά του σημεία είναι ότι μπορεί να μοιράσει τις εργασίες σε μια τοπολογία δικτύου.

4. Theano

Το Theano είναι ένα Βαθιάς Μάθησης περιβάλλον, πήρε το όνομα του από την σύζυγο του Θαλή (Αρχαίος Έλληνας μαθηματικός και φιλόσοφος). Μπορεί και υλοποιεί αλγορίθμους εκμάθησης τόσο σε CPU όσο και GPU. Υποστηρίζει τύπους δεδομένων όπως πολυδιάστατων πινάκων, καθώς και τύπου δεδομένων Tensor. Ο Tensor τύπος δεδομένων θα δούμε παρακάτω ότι είναι ό βασικός τύπος της γλώσσας προγραμματισμού LUA. Φαίνεται παρακάτω ότι η υλοποίηση και η ολοκλήρωση Βαθιάς Μηχανικής Μάθησης μπορεί και αποδίδει καλύτερα σε χρόνο όπως στην παρακάτω εικόνα.

5.Keras

Το Keras, είναι μια βιλιοθήκη καθαρά γραμμένη σε Python. Μπορεί και τρέχει στο ανώτερο επίπεδο των Tensorflow, CNTK και Theano, και χρησιμοποιείται ευρέως για γρήγορα αποτελέσματα με πολύ λίγο κώδικα. Μπορεί και τρέχει πολύ εύκολα σε ένα σύστημα με ή χωρίς ενεργοποιημένη GPU.

Ένα σημαντικό μειονέκτημα είναι ότι δεν έχει αρκετούς αλγόριθμους υλοποιημένους.

6.Torch & LUA

Το Torch είναι ένα επιστημονικό πλαίσιο εργασίας υπολογιστών που προσφέρει ευρεία υποστήριξη στους αλγόριθμους Μηχανικής Μάθησης. Πρόκειται για ένα βαθύ πλαίσιο μάθησης βασισμένο σε Lua και χρησιμοποιείται ευρέως, όπως Facebook, Twitter και Google. Χρησιμοποιεί το CUDA μαζί με τις βιβλιοθήκες C / C ++ για επεξεργασία και βασικά έγινε για να κλιμακώσει την παραγωγή μοντέλων κτιρίων και να προσφέρει γενική ευελιξία. Το περιβάλλον αυτό έχει ως γλώσσα προγραμματισμού την Lua. Αναφέρουμε περισσότερα για αυτή τη γλώσσα παρακάτω είναι πολύ γρήγορη και υποστηρίζει τύπο δεδομένων Tensor. Μπορεί και αυτή να εκτελεστεί με ή χωρίς GPU και έχει μεγάλη γκάμα αλγορίθμων υλοποιημένων μέσω των βιβλιοθηκών της nn για χρήση CPU , και cunn για χρήση GPU. Όλες οι βιβλιοθήκες μπορούν να εγκατασταθούν με την Luarocks όπως ακριβώς χρησιμοποιούμε την pip για να εγκαταστήσουμε βιβλιοθήκες στην Python.

$ luarocks install nn

Η Lua σημαίνει φεγγάρι και είναι μια scripting γλώσσα προγραμματισμού, ισχυρή, αποτελεσματική ελαφριά και μπορεί να ενσωματωθεί σε οποιοδήποτε υπολογιστικό σύστημα όπου μπορεί και τρέχει ένα C compiler. Έχει χρησιμοποιηθεί εκτενέστερα στο προγραμματισμό παιχνιδιών. Η Lua έχει την φήμη της πιο γρήγορης γλώσσας scripting, και μπορεί να εκτελεστεί σε περιβάλλον Linux, Windows ακόμα και σε κινητές συσκευές σε επεξεργαστές ARM & Rabbit. Διαθέτει αυτόματη διαχείριση μνήμης με αυξημένη συλλογή διαχείριση απορριμμάτων (carbage collector).

Είναι open source με περίπου 24.000 γραμμές

Μία γρήγορη εγκατάσταση Lua είναι η παρακάτω σε ένα τερματικό

curl -R -O http://www.lua.org/ftp/lua-5.3.5.tar.gz

```
tar zxf lua-5.3.5.tar.gz

cd lua-5.3.5

make linux test
```

Μαζί με την Lua θα εγκαταστήσουμε το framework Torh. Το Torch (A SCIENTIFIC COMPUTING FRAMEWORK FOR LUAJIT) ή άλλως ένα επιστημονικό περιβάλλον πληροφόρησης για την LuaGit έχει δυνατότητες όπως :

Ισχυρή διάταξη N-διαστάσεων

Πολλές ρουτίνες για την ευρετηρίαση, τον τεμαχισμό, τη μεταφορά, ...

Καταπληκτική διασύνδεση με C, μέσω του LuaJIT

Γραμμικές ρουτίνες άλγεβρας

Νευρωνικό δίκτυο και ενεργειακά μοντέλα

Αριθμητικές ρουτίνες βελτιστοποίησης

Γρήγορη και αποδοτική υποστήριξη GPU

Ενσωματωμένο, με θύρες σε iOS και Android backends

Μέχρι αργά, το PyTorch έχει δει ένα υψηλό επίπεδο υιοθεσίας στο πλαίσιο της κοινότητας πλαισίου Βαθιάς Μάθησης και θεωρείται ανταγωνιστής του TensorFlow. Το PyTorch είναι βασικά ένα λιμάνι στο πλαίσιο βαθιάς μάθησης Torch που χρησιμοποιείται για την κατασκευή νευρωνικών δικτύων και για την εκτέλεση υπολογισμών τανυστή που είναι μεγάλοι όσον αφορά την πολυπλοκότητα.

Σε αντίθεση με το Torch, το PyTorch τρέχει στην Python, πράγμα που σημαίνει ότι όποιος έχει βασική κατανόηση της Python μπορεί να ξεκινήσει να κατασκευάζει τα δικά του Βαθιά μοντέλα Μάθησης.

Δεδομένου του αρχιτεκτονικού στυλ του PyTorch πλαισίου, όλη η διαδικασία βαθιάς μοντελοποίησης είναι πολύ πιο απλή και διαφανής σε σύγκριση με τον φακό.

7.MXNet

Σχεδιασμένο ειδικά για τους σκοπούς της υψηλής απόδοσης, της παραγωγικότητας και της ευελιξίας, το MXNet (προφέρεται ως mix-net) είναι ένα πλαίσιο Βαθιάς Μάθησης που υποστηρίζεται σε γλώσσες Python, R, C ++ και Julia.

Η ομορφιά του MXNet είναι ότι δίνει στον χρήστη τη δυνατότητα να κωδικοποιεί σε διάφορες γλώσσες προγραμματισμού. Αυτό σημαίνει ότι μπορεί να εκπαιδεύσετε σε Βαθιά Μάθηση με οποιαδήποτε γλώσσα. Είναι γραμμένο σε C ++ και CUDA, το MXNet είναι σε θέση να κλιμακωθεί και να λειτουργήσει με μια πλειάδα GPU, γεγονός που το καθιστά απαραίτητο για τις επιχειρήσεις. Η Amazon χρησιμοποίησε το MXNet ως βιβλιοθήκη αναφοράς για Βαθιά Μάθηση.

Το MXNet υποστηρίζει δίκτυα μακράς βραχυπρόθεσμης μνήμης (LTSM) μαζί με RNN και CNN.

Αυτό το βαθύ πλαίσιο μάθησης είναι γνωστό για τις ικανότητές του στην απεικόνιση, την αναγνώριση χειρογράφου / ομιλίας, την πρόβλεψη και το NLP (Natural Language Processing).

8.OPENMPI

Εκτός όμως από την κάρτα γραφικών μπορούμε με δωρεάν λογισμικό να τρέξουμε οποιοδήποτε κώδικα, οποιασδήποτε γλώσσας προγραμματισμού παράλληλα σε όλους τους βασικού πυρήνες ενός επεξεργαστή. Η χρήση λογισμικού με όνομα OPENMPI δίνει τη δυνατότητα αναγνωρίζοντας το υλικό του υπολογιστή να μπορεί να εκτελέσει τον κώδικα σε διαφορετικά νήματα αφήνοντας στο λειτουργικό σύστημα τον διαμερισμό των εργασιών αυτών.

Η βιβλιοθήκη openmpi μπορεί να ενσωματωθεί και σε γλώσσα python όπου με ένα απλό import της αντίστοιχης βιβλιοθήκης μπορούμε να έχουμε την δύναμη όλων των πυρήνων στη δυνατότητα μας.

Η εγκατάσταση σε περιβάλλον ή αλλιώς build αφού έχουμε τον πηγαίο κώδικα θα προσπαθήσουμε να εκτελέσουμε την εγκατάσταση με τις παρακάτω εντολές σε περιβάλλον Linux

shell$ gunzip -c openmpi-3.1.2.tar.gz | tar xf -

shell$ cd openmpi-3.1.2

shell$ ./configure --prefix=/usr/local

shell$ make all install

Σημείωση για χρήση με cuda (πυρήνες κάρτας γραφικών):

Για να χρησιμοποιήσουμε το openmpi σε περιβάλλον cuda, θα πρέπει να ενημερώσουμε το αρχείο configure με τη διεύθυνση του εκτελέσιμου αρχείου cuda, καθώς και την διαδρομή του δίσκου με τις αντίστοιχες βιβλιοθήκες.

9. SPARK

Ένα άλλο framework που δοκιμάσαμε είναι το spark της Apache. Το spark είναι ένα σύνολο εντολών όπου τρέχει σε περιβάλλον linux σε χαμηλό επίπεδο εντολών λόγω του ότι παρατηρήθηκε ότι όσο πιο κοντά στο λειτουργικό εκτελούμε διάφορες διαδικασίες τόσο και ταχύτερα εκτελούνται αυτές. Ένα παράδειγμα είναι η ένωση κάποιων δεδομένων. Θα δούμε παρακάτω ότι η μορφή των δεδομένων μπορεί να σε μορφή κειμένου.

ΕΓΚΑΤΑΣΤΑΣΗ

Παραδείγματος χάρη οι τιμές σε ένα χρηματιστήριο μπορεί να είναι σε ένα txt αρχείο της παρακάτω μορφής:

451.12 451.06 451.0 450.98 450.96 450.92 450.78 450.64 450.62 450.5 450.48 450.47 450.46 450.34 450.16 450.15 450.12 450.1 450.0 449.8 449.78 449.7 449.39 449.36 449.35 449.26 449.21 449.18 449.15 449.07 448.97 448.96 448.87 448.77 448.63 448.62 448.6 448.58 448.5 448.49 448.42 448.41 448.32 448.31 448.2 448.19 448.0 447.96 447.89 447.86 447.84 447.83 447.82 447.68 447.67 447.53 447.52 447.43 447.4 447.39 451.35 451.12 451.06 451.0 450.98 450.96 450.92 450.78 450.64 450.62 450.5 450.48 450.47 450.46 450.34 450.16 450.15 450.12 450.1 450.0 449.8 449.78 449.7 449.39 449.36 449.35 449.26 449.21 449.18 449.15 449.07 448.97 448.96 448.87 448.77 448.63 448.62 448.6 448.58 448.5 448.49 448.42 448.41 448.32 448.31 448.2 448.19 448.0 447.96 447.89 447.86 447.84 447.82 447.68 447.67 447.53 447.52 447.43 447.4 447.39 451.35 451.12 451.06 451.0 450.98 450.96 450.92 450.78 450.64 450.62 450.5 450.48 450.47 450.46 450.34 450.16 450.15 450.12 450.1 450.0 449.8 449.78 449.7 449.39 449.36 449.35 449.26 449.21 449.18 449.15 …

Είναι συνήθης μορφή στα δεδομένα χρονοσειρών, επίσης το γρηγορότερο διάβασμα είναι σε μορφή txt. Υπάρχουν σε αρκετές γλώσσες ειδικές συναρτήσεις όπου διαβάζουν αυτές τις τιμές και τις φορτώνουν κατευθείαν στην μνήμη για επεξεργασία πολύ ταχύτερα τουλάχιστον από τις βάσεις δεδομένων.

Το framework Spark μπορεί να εκτελέσει τοπικά με παρόμοιο τρόπο ότι ακριβώς κάνει και το openmpi. Όμως αυτό που το κάνει αρκετά ισχυρό σε επίπεδο επεξεργασίας, είναι ότι μπορεί να ενορχηστρώνει τις διεργασίες αυτές σε επίπεδο δικτύου με διαφορετικά υπολογιστικά συστήματα. Δηλαδή σε ένα δίκτυο με αρκετούς υπολογιστές , στο ίδιο δίκτυο για να επικοινωνούν μεταξύ τους μπορεί ένα υπολογιστής ο master, όπου εκτελείται και το πρόγραμμα , δίνει εντολή σε όλους τους άλλους να επεξεργαστούν μέρος του αρχείου prices.txt. Πρακτικά είναι σαν να ορίζει την επανάληψη όλων των τιμών του αρχείου ανάλογα τους slaves (οι υπόλοιποι υπολογιστές) π.χ. ο πρώτος θα διαβάσει το αρχείο όλο και θα εκτελέσει την από 1 - 1000, ο δεύτερος στον ίδιο χρόνο θα διαβάσει όλο το αρχείο και θα εκτελέσει την λούπα από την τιμή 1001 – 2000 . Αφού τελειώσουν όλες οι διαδικασίες αναφέρονται στο master, ο οποίος αποδίδει το αποτέλεσμα συνολικά. Η παραπάνω τεχνική έχει ένα μειονέκτημα και αυτό είναι ο χρόνος διαβάσματος του αρχείου στη μνήμη. Να σημειώσουμε εδώ ότι το Spark μπορεί να με σύστημα αρχείων HDFS (HADOOP Distributed File System) το οποίο λειτουργεί ώς ένα σύσημα

αρχείων, δεν υποστηρίζονται συντομεύσεις ή άλλες εφαρμογές. Αλλά μπορεί να αντιγράφει τα αρχεία πολύ αποτελεσματικά.

Το HDFS έχει σχεδιαστεί για να αποθηκεύει αξιόπιστα πολύ μεγάλα αρχεία σε μηχανές μεγάλου συμπλέγματος (clusters). Η διαδρομή δίσκου είναι δυκτιακή όπως ενός fileserver.

192.168.1.45/folder/subdolder/data/data0.txt

Αποθηκεύει κάθε αρχείο ως ακολουθία μπλοκ. όλα τα μπλοκ σε ένα αρχείο εκτός από το τελευταίο μπλοκ έχουν το ίδιο μέγεθος. Τα μπλοκ ενός αρχείου αντιγράφονται για ανοχή σφάλματος. Το μέγεθος του μπλοκ και ο παράγοντας αναπαραγωγής μπορούν να διαμορφωθούν ανά αρχείο. Μια εφαρμογή μπορεί να καθορίσει τον αριθμό αντιγράφων ενός αρχείου.

Ο παράγοντας αναπαραγωγής μπορεί να οριστεί κατά το χρόνο δημιουργίας αρχείου και μπορεί να αλλάξει αργότερα. Τα αρχεία σε HDFS γράφονται μία φορά και έχουν αυστηρά έναν συγγραφέα ανά πάσα στιγμή. Άρα εδώ ένα άλλο μειονέκτημα είναι ότι χρειάζεται να αντιγράφουμε τα αρχεία για κάθε επεξεργαστή που θέλει να τρέξει ένα κομμάτι από αυτά τα δεδομένα. Σίγουρα βέβαια είναι ένας από τους ασφαλέστερους τρόπου να μπορεί να εκτελεστεί σε πραγματικά μεγάλα δεδομένα με ασφάλεια και αξιοπιστία.

Σε αυτό το σημείο θα πρέπει να αναφέρουμε ότι

Η ιεραρχική μορφή δεδομένων (HDF) είναι ένα σύνολο μορφών αρχείων (HDF4, HDF5) που έχουν σχεδιαστεί για την αποθήκευση και την οργάνωση μεγάλων ποσοτήτων δεδομένων. Αρχικά αναπτύχθηκε στο Εθνικό Κέντρο Υπολογιστικών Εφαρμογών, υποστηρίζεται από τον Όμιλο HDF, μια μη κερδοσκοπική εταιρεία με αποστολή να εξασφαλίσει τη συνεχή ανάπτυξη των τεχνολογιών HDF5 και τη συνεχή πρόσβαση στα δεδομένα που είναι αποθηκευμένα σε HDF.

Σύμφωνα με αυτόν τον στόχο, οι βιβλιοθήκες HDF και τα σχετικά εργαλεία είναι διαθέσιμα υπό μια φιλελεύθερη άδεια τύπου BSD για γενική χρήση. Το HDF υποστηρίζεται από πολλές εμπορικές και μη εμπορικές πλατφόρμες λογισμικού, συμπεριλαμβανομένων των Java, MATLAB, Scilab, Octave, Mathematica, IDL, Python, R και Julia. Η ελεύθερη διανομή HDF αποτελείται από τη βιβλιοθήκη, βοηθητικά προγράμματα γραμμής εντολών, πηγή δοκιμαστικής σουίτας, διεπαφή Java και HDF Viewer με βάση το Java.

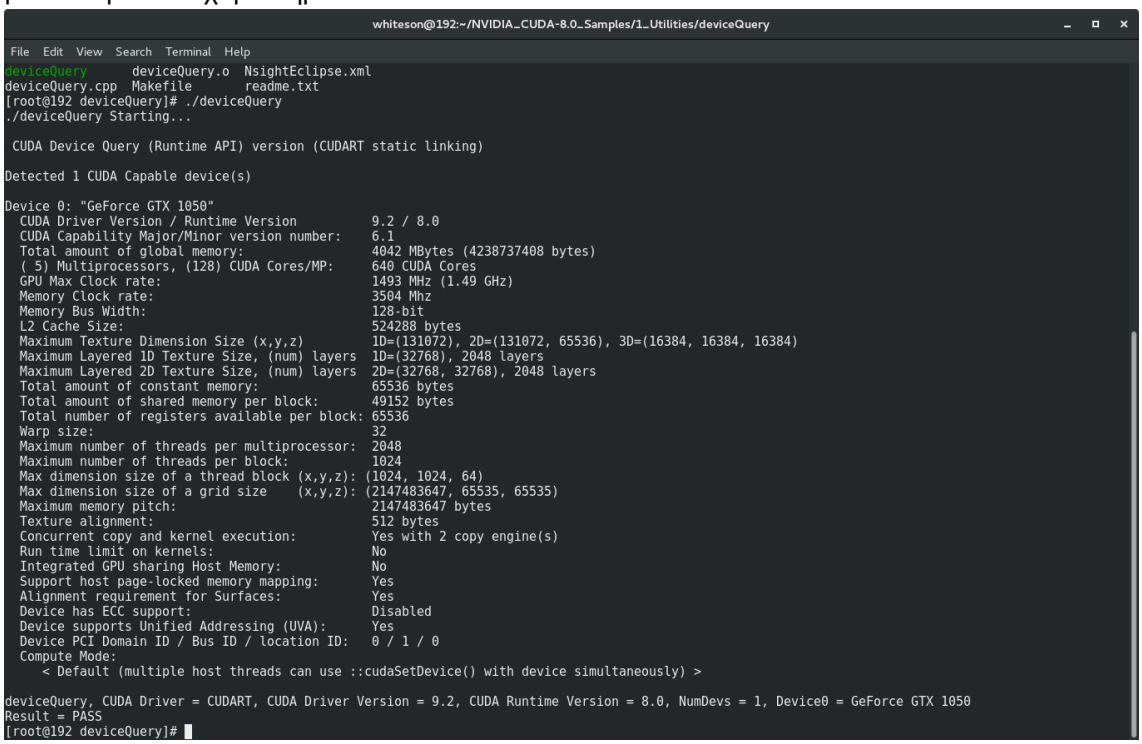
Σημείωση ένας αριθμός τύπου float χρειάζεται παραπάνω μνήμη απ' ότι χρειάζεται σε txt μορφή. Άρα κατά την υλοποίηση και εισαγωγή σε ένα αλγόριθμό αντί να σχηματίσουμε ένα πίνακα με float τιμές, κάνουμε προσπέλαση αρχείου και πριν τον υπολογισμό τις εκάστοτε πράξης κάνουμε casting στην μεταβλητή όπου έχει τον αριθμό σε αλφαριθμητική μορφή σε αυτή με float.

1.epoch ορίζεται ώς ο κύκλος μιας εκπαίδευσης αλγορίθμου όπου αφού εξεταστεί το στατιστικό λάθος περνάει στο επόμενο epoch για την ελαχιστοποίηση του σφάλματος.

4. ΠΕΡΙΒΑΛΛΟΝ ΕΡΓΑΣΙΑΣ

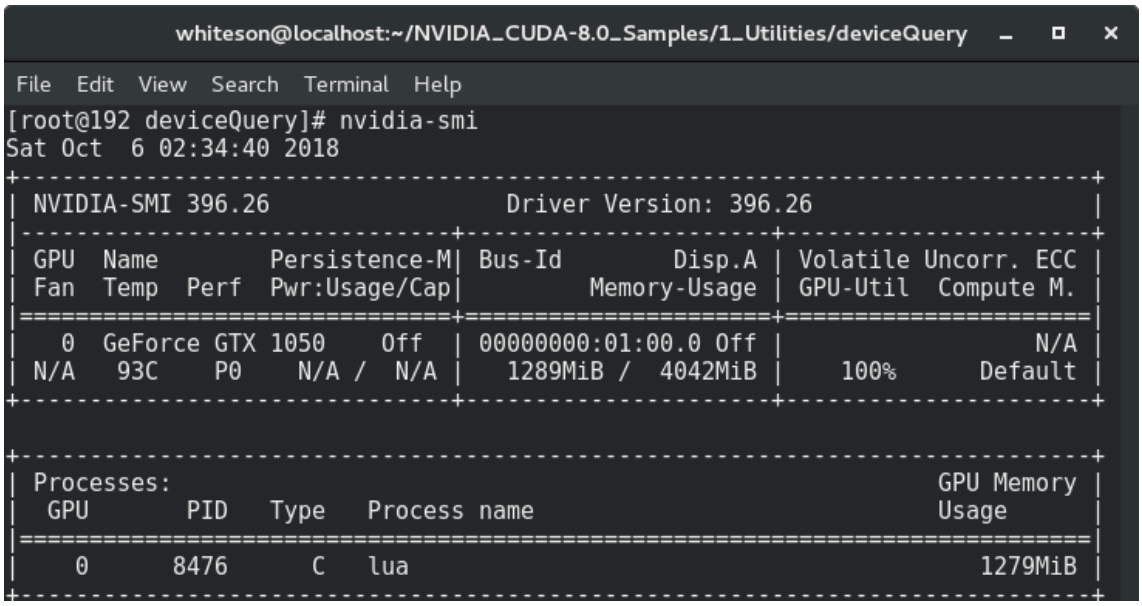
4.1 ΥΠΟΛΟΓΙΣΤΙΚΌ ΣΥΣΤΗΜΑ ΚΑΙ ΛΟΓΙΣΜΙΚΟ

Τα πειράματα και η εκτέλεση γίναν σε ένα φορητό υπολογιστή με κάρτα γραφικών Nvidia 1050 με τα παρακάτω χαρακτηριστικά.



DEVICE QUERY COMMAND

Επίσης με την παρακάτω εντολή μπορούμε να δούμε τις διεργασίες που τρέχουν καθώς και την μνήμη της κάρτας γραφικών που χρησιμοποιούν.



NVIDIA-SMI

Μία υψηλού γλώσσα προγραμματισμού, αφενός προσφέρει πολύ γρήγορη υλοποίηση όμως τις περισσότερες φορές χάνει σε απόδοση και ταχύτητα σε σχέση με αυτές που χρησιμοποιούν C ,C++. Στην ενότητα 5 θα αναλύσουμε και θα δούμε συγκριτικά αποτελέσματα απόδοσης των εν λόγω διαφορών. Το λογισμικό που χρησιμοποιήσαμε ήταν για λειτουργικό σύστημα RedHat την έκδοση για developers, μπορεί να αποκτηθεί με μια δωρεάν εγγραφή στην ιστοσελίδα τους. Επιλέξαμε Red Hat λόγω του ότι ενώ αυτή η έκδοση προσφέρει λιγότερη συνδεσιμότητα, όμως είναι ένα πολύ πιο σταθερό περιβάλλον εργασίας, με καλύτερη ασφάλεια στην χρήση των βιβλιοθηκών που χρησιμοποιεί. Για την εκμάθηση των αλγορίθμων ξεκινήσαμε πρώτα με το SPARK της Apache. Το framework δεν είχε ωριμάσει αρκετά για να μας προσφέρει μια λύση η χρήση αλγορίθμων Sparse AutoEncoder ή Convolutional Autoencoder, και δεν τον επιλέξαμε λόγω του ότι το SPARK μπορεί και εκτελείται σε 2 modes, local ή netowrk. Τοπικά προσφέρει παράλληλους υπολογισμούς με κύριο πλεονέκτημα ότι εκτελεί τις πράξεις στην μνήμη του υπολογιστή και συνδέεται εύκολα με clusters, δημιουργώντας αντιγραφή των δεδομένων, με εμφανή επίπτωση στη ταχύτητα. Ο δεύτερος τρόπος είναι μέσω δικτύου. Σε αυτή την εκτέλεση που δεν μπορούσαμε να δοκιμάσουμε λόγω υλικών πόρων π.χ. χρειαζόμαστε ένα δίκτυο από υπολογιστικές μονάδες. Η μια μονάδα έχει το ρόλο του master, και οι άλλου οι slaves. Ο master τρεχει τον αλγόριθμο, αντιγράφει τα αρχεία ώστε να είναι διαθέσιμα στους slaves και όταν οι slaves ολοκληρώσουν την εκτέλεση παραδίδουν τα αποτελέσματα στον master ο οποίος αποδίδει το τελικό αποτέλεσμα. Οι βασικές μονάδες δεδομένων στο Spark ονομάζονται Εύκαμπτα Κατανεμημένα Σύνολα Δεδομένων - Resilient Distributed Datasets ή (RDDs). Πρόκειται για μια κατανεμημένη, αμετάβλητη και ανεκτική σε λάθη αφαίρεση μνήμης που συλλέγει ένα σύνολο στοιχείων στα οποία μπορεί να εφαρμοστεί ένα σύνολο λειτουργιών είτε για την παραγωγή άλλων RDD (μετασχηματισμοί) είτε για επιστροφές (δράσεις). Τα RDD μπορούν να βρίσκονται στη μνήμη, στο δίσκο ή σε συνδυασμό. Ωστόσο, υπολογίζονται μόνο σε

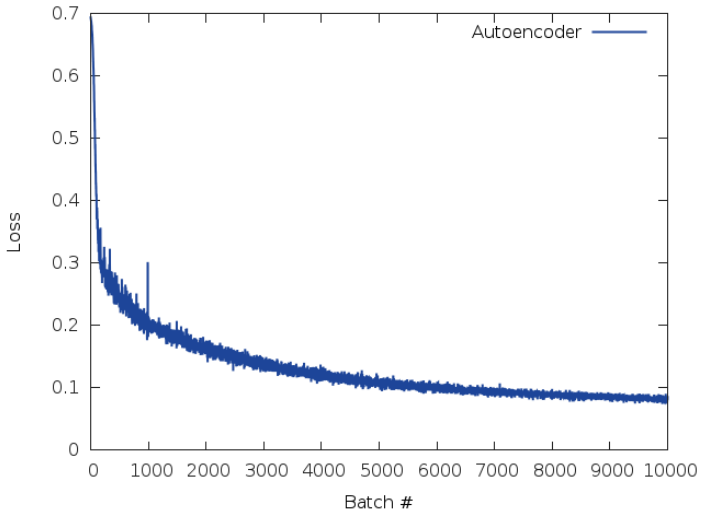
ενέργειες μετά από μια στρατηγική οκνηρής αποτίμησης Lazy Evaluation (LE), προκειμένου να εκτελούνται ελάχιστοι υπολογισμοί και να αποτρέπεται η περιττή χρήση της μνήμης. Τα RDDs δεν αποθηκεύονται στη μνήμη από προεπιλογή, επομένως, όταν τα δεδομένα επαναχρησιμοποιούνται, απαιτείται μια μέθοδος διατήρησης για να αποφευχθεί η ανασυγκρότηση. Το Open Mpi μπορεί και χρησιμοποιεί κοινή μνήμη κατά την εκτέλεση του και μπορεί και τρέχει 1 script όπως είδαμε ταυτόχρονα σε πολλούς πυρήνες, γι' αυτό το λόγο είναι shared η μνήμη του. Το OpenMpi μας βοήθησε να τρέξουμε παράλληλα σε όλα τα αρχεία σε όλους τους πηρύνες της μητρικής, να διαβάσουμε τα mat αρχεία σε περιβάλλον python, και να εξάγουμε σε txt μορφή όλους τους πίνακας δημιουργώντας ουρές για τις 60 στήλες του βάθους τις τιμής. Η γλώσσα προγραμματισμού Lua χρησιμοποιείται μπορεί να εκτελεστεί σε όλα τα υπολογιστικά συστήματα που μπορούν να κάνουν compile γλώσσα ANSI C. Είναι πολύ δυνατή γλώσσα προγραμματισμού καθώς μπορεί να εκτελεστεί γρήγορα και έχει πολύ ελαφρύ αποτύπωμα σε ένα υπολογιστικό σύστημα. Έχει χρησιμοποιηθεί σε πολλά περιβάλλοντα εργασίας και σε παιχνίδια. Αφού εγκαταστήσαμε την lua μπορούμε να χρησιμοποιούμε το περιβάλλον εργασίας Torch. Το περιβάλλον είναι γραμμένο σε C με έτοιμες συναρτήσεις σε CUDA. Για την ορθή χρήση της κάρτας γραφικών στο περιβάλλον πρέπει να οριστούν στο configuration το make αρχείου τα σωστά paths των βιβλιοθηκών της Nvidia. Όλες οι εγκαταστάσεις εκτός της ειδικής εγκατάστασης της NVIDIA, περιλαμβάνει την αποθήκευση πηγαίου κώδικα στο υπολογιστή μας χρήση εντολών configure, make && make install, σύνδεση με τα αντίστοιχα paths των βιβλιοθηκών όπως σε περιβάλλον linux, ορίζονται με της System Variables του λειτουργικού μας συστήματος PATH, LD_LIBRARY όπου με την εντολή export όρίζονται.

5.ΕΦΑΡΜΟΓΕΣ & ΠΑΡΑΔΕΙΓΜΑΤΑ

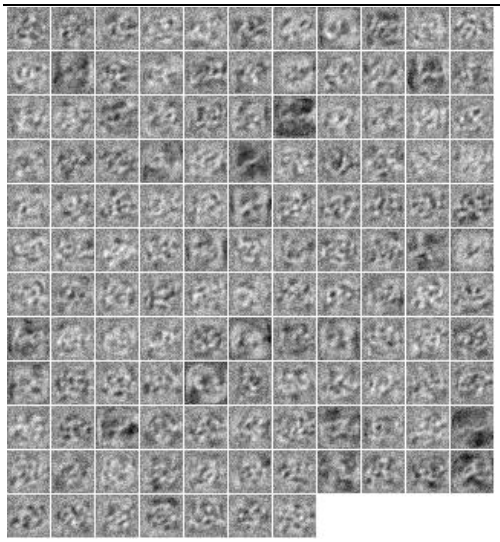
5.1 AutoEncoder

```lua
local nn = require 'nn'

local Model = {
  features = 128
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, self.features))
  self.encoder:add(nn.ReLU(true))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.features, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model

```

Lua source code



Training Autoencoder AE



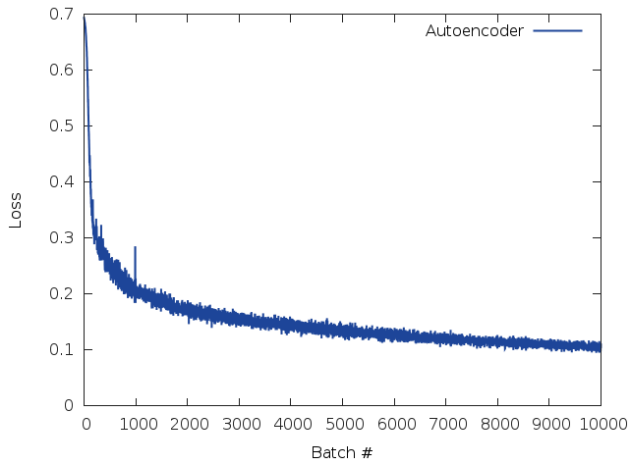
AE WEIGHTS

AE reconstructions

5.2 Sparse autoencoder

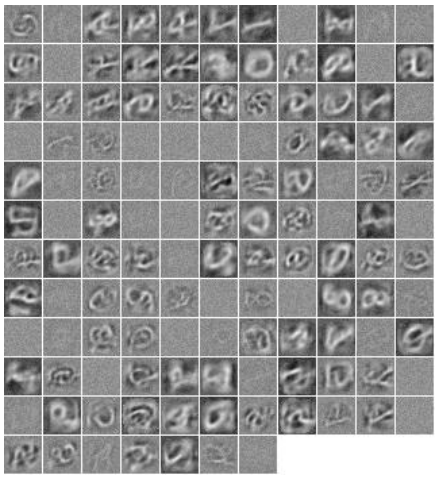
```lua
local nn = require 'nn'

local Model = {
  features = 128
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, self.features))
  self.encoder:add(nn.ReLU(true))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.features, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(nn.L1Penalty(1e-5)) -- L1 penalty on activations
  self.autoencoder:add(self.decoder)
end

return Model
```

Lua source Code



Sparse AE Training

Sparse AE Reconstructions

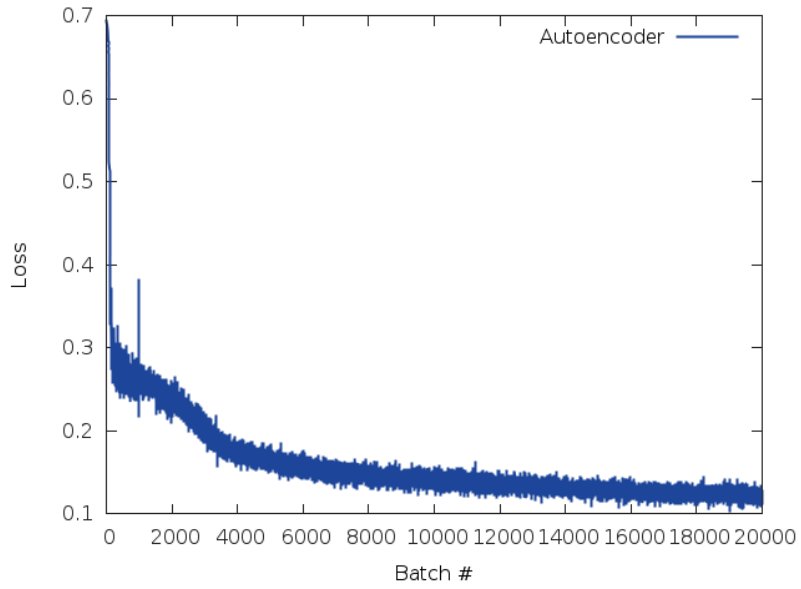


Sparse AE Weights

5.3 DeepAutoencoder

```lua
local nn = require 'nn'

local Model = {}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, 128))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(128, 64))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(64, 32))
  self.encoder:add(nn.ReLU(true))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(32, 64))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(64, 128))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(128, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model

```

Lua source Code Deep Autoencoder



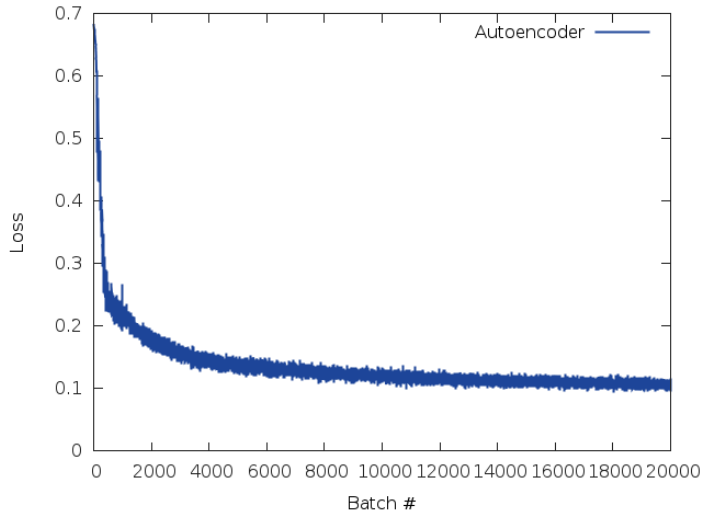
Deep ae training

Deep ae reconstructions

5.4 Convolution Autoencoder

```lua
local nn = require 'nn'

local Model = {}

function Model:createAutoencoder(X)
  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, 1, X:size(2), X:size(3)))
  self.encoder:add(nn.SpatialConvolution(1, 16, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2, 1, 1))
  self.encoder:add(nn.SpatialConvolution(16, 8, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2, 1, 1))
  self.encoder:add(nn.SpatialConvolution(8, 8, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.SpatialConvolution(8, 8, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialUpSamplingNearest(2))
  self.decoder:add(nn.SpatialConvolution(8, 8, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialUpSamplingNearest(2))
  self.decoder:add(nn.SpatialConvolution(8, 16, 3, 3, 1, 1, 0, 0))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialUpSamplingNearest(2))
  self.decoder:add(nn.SpatialConvolution(16, 1, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model

```

Lua model source code



ConvAE Traing

ConvAEreconstructions

5.5 UpconvolutionalAutoEncoder

```lua
local nn = require 'nn'

local Model = {}

function Model:createAutoencoder(X)
  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, 1, X:size(2), X:size(3)))
  self.encoder:add(nn.SpatialConvolution(1, 16, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2, 1, 1))
  self.encoder:add(nn.SpatialConvolution(16, 8, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2, 1, 1))
  self.encoder:add(nn.SpatialConvolution(8, 8, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.SpatialFullConvolution(8, 8, 4, 4, 2, 2, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialFullConvolution(8, 16, 4, 4, 2, 2, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialFullConvolution(16, 1, 4, 4, 2, 2, 3, 3))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model
```

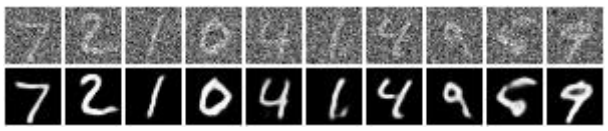
Model

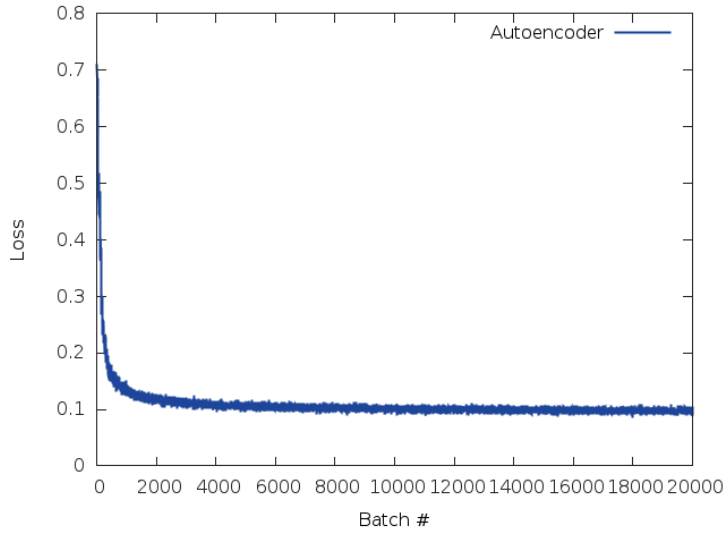
Reconstractions

5.6 Denoising Autoencoder

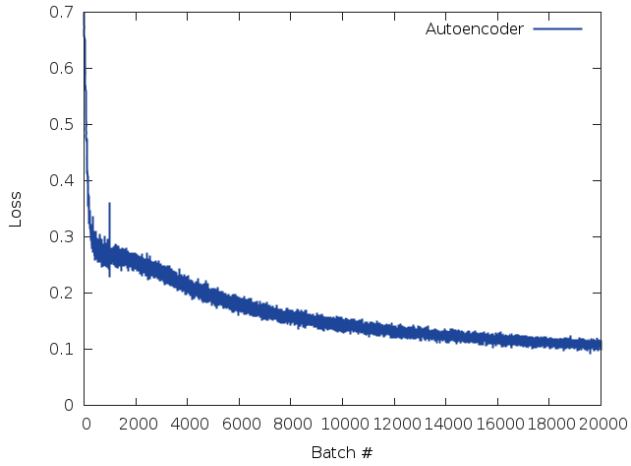
```lua
local nn = require 'nn'
require 'dpnn'

local Model = {}

function Model:createAutoencoder(X)
  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, 1, X:size(2), X:size(3)))
  self.encoder:add(nn.SpatialConvolution(1, 32, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2, 1, 1))
  self.encoder:add(nn.SpatialConvolution(32, 32, 3, 3, 1, 1, 1, 1))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.SpatialMaxPooling(2, 2, 2, 2))

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.SpatialConvolution(32, 32, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialUpSamplingNearest(2))
  self.decoder:add(nn.SpatialConvolution(32, 32, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.SpatialUpSamplingNearest(2))
  self.decoder:add(nn.SpatialConvolution(32, 1, 3, 3, 1, 1, 1, 1))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.noiser = nn.WhiteNoise(0, 0.5) -- Add white noise to inputs during training
  self.autoencoder:add(self.noiser)
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model

```

Model





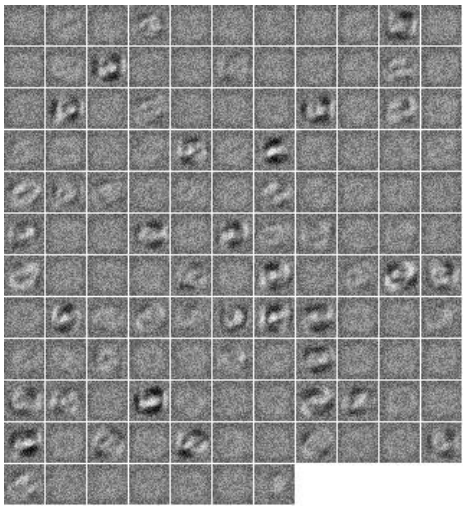
CAE



CAE TRAINING

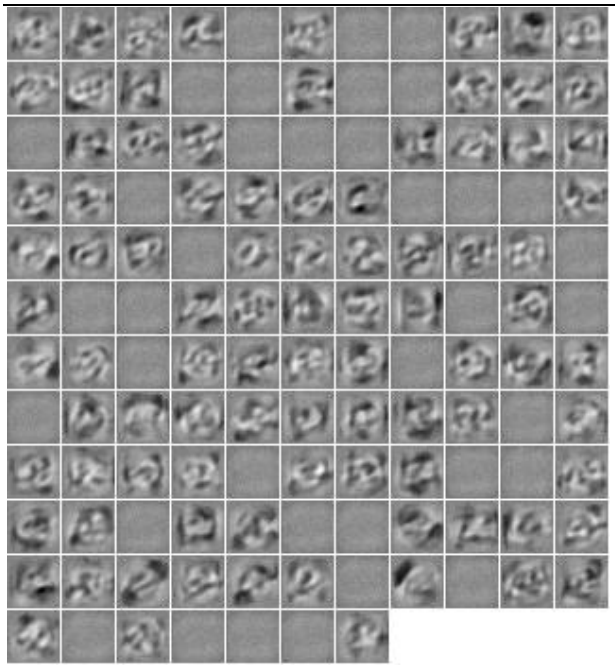
CAE RECONSTRUCTIONS

Caeweights

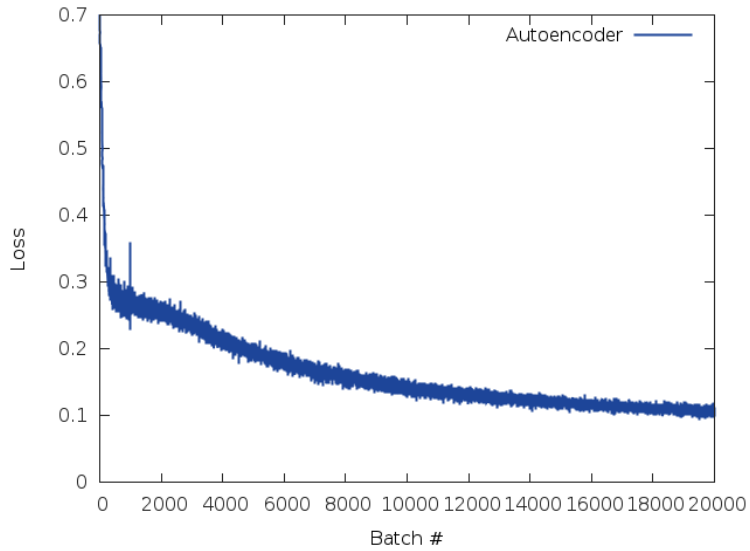


Το παραπάνω το τρέξαμε με μια επανάληψη, όμως όταν χρησιμοποιήσαμε 20 επαναλήψεις είδαμε ότι ο αλγόριθμος δούλεψε πολύ καλύτερα με συνάρτηση λάθους, και τα βάρη είναι πιο καθαρά.

CAE – RECONSTRUCTION – 20 epochs



CAE – WEIGHTS – 20 epochs



CAE – TRAINING – 20 epochs

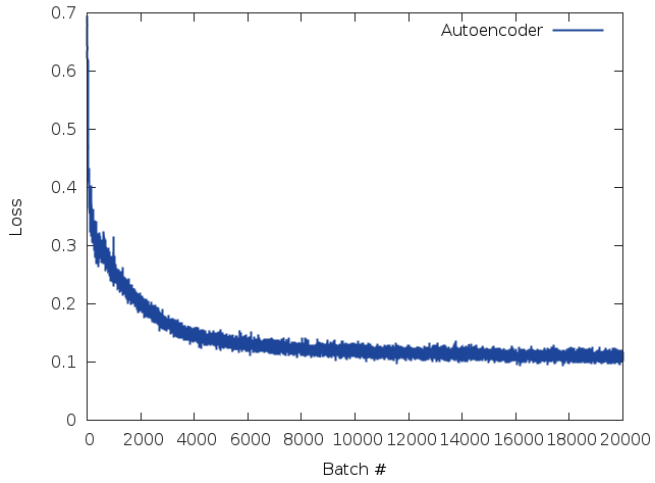
5.7Seq2seqAE

```lua
local nn = require 'nn'
require 'rnn'

local Model = {
  cellSizes = {256, 256}, -- Number of LSTM cells
  encLSTMs = {},
  decLSTMs = {}
}

-- Copy encoder cell and output to decoder LSTM
function Model:forwardConnect()
  for l = 1, #Model.decLSTMs do
    Model.decLSTMs[l].userPrevOutput = Model.encLSTMs[l].output[self.seqLen]
    Model.decLSTMs[l].userPrevCell = Model.encLSTMs[l].cell[self.seqLen]
  end
end

-- Copy decoder gradients to encoder LSTM
function Model:backwardConnect()
  for l = 1, #Model.encLSTMs do
    Model.encLSTMs[l].userNextGradCell = Model.decLSTMs[l].userGradPrevCell
    Model.encLSTMs[l].gradPrevOutput = Model.decLSTMs[l].userGradPrevOutput
  end
end

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)
  self.seqLen = X:size(2) -- Treat rows as a sequence

  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.Transpose({1, 2})) -- Transpose to seqlen x batch
  for l = 1, #Model.cellSizes do
    local inputSize = l == 1 and X:size(3) or Model.cellSizes[l - 1]
    self.encLSTMs[l] = nn.SeqLSTM(inputSize, Model.cellSizes[l])
    self.encoder:add(self.encLSTMs[l])
  end

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Transpose({1, 2})) -- Transpose to seqlen x batch
  for l = 1, #Model.cellSizes do
    local inputSize = l == 1 and X:size(3) or Model.cellSizes[l - 1]
    self.decLSTMs[l] = nn.SeqLSTM(inputSize, Model.cellSizes[l]):remember('eval') -- Retain hidden state on consecutive calls to forward during evaluation
    self.decoder:add(self.decLSTMs[l])
  end
  self.decoder:add(nn.Sequencer(nn.Linear(Model.cellSizes[#Model.cellSizes], X:size(3)))) -- Reconstruct columns
  self.decoder:add(nn.Transpose({1, 2})) -- Transpose back to batch x seqlen
  self.decoder:add(nn.Sigmoid(true))

  -- Create dummy container for getParameters (no other way to combine storage pointers)
  self.dummyContainer = nn.Sequential()
  self.dummyContainer:add(self.encoder)
  self.dummyContainer:add(self.decoder)

  -- Create autoencoder wrapper
  self.autoencoder = {
    parent = self
```

Seg2Seg model

```lua
    -- Create autoencoder wrapper
    self.autoencoder = {
      parent = self
    }

    -- Create CUDA wrapper
    function self.autoencoder:cuda()
      self.parent.encoder:cuda()
      self.parent.decoder:cuda()
    end

    -- Create replace wrapper
    function self.autoencoder:replace(fn)
      self.parent.dummyContainer:replace(fn)
    end

    -- Create getParameters wrapper
    function self.autoencoder:getParameters()
      return self.parent.dummyContainer:getParameters()
    end

    -- Create training wrapper
    function self.autoencoder:training()
      self.parent.encoder:training()
      self.parent.decoder:training()
    end

    -- Create evaluate wrapper
    function self.autoencoder:evaluate()
      self.parent.encoder:evaluate()
      self.parent.decoder:evaluate()
    end
  end

  -- Create forward wrapper
  function self.autoencoder:forward(x)
    local encOut = self.parent.encoder:forward(x)
    self.parent:forwardConnect()

    -- Use target vector in training, sample from self in evaluate
    if self.parent.decoder.train then
      -- Shift decoder input sequence by one step forward
      local decInSeq = x:clone()
      decInSeq[{{}, {2, x:size(2)}, {}}] = decInSeq[{{}, {1, x:size(2) - 1}, {}}]
      decInSeq[{{}, {1}, {}}]:zero() -- Start from vector of zeros
      return self.parent.decoder:forward(decInSeq)
    else
      local decOut = torch.zeros(x:size()):typeAs(x)
      local decOutPartial = torch.zeros(x:size(1), 1, x:size(3)):typeAs(x) -- Start from vector of zeros

      -- Feed outputs back into self
      self.parent.decoder:forget() -- Clear hidden state
      for t = 1, self.parent.seqLen do
        decOutPartial = self.parent.decoder:forward(decOutPartial)
        decOut[{{}, {t}, {}}] = decOutPartial
      end
      return decOut
    end
  end

  -- Create backward wrapper
  function self.autoencoder:backward(x, gradLoss)
    -- Shift decoder input sequence by one step forward
    local decInSeq = x:clone()
    decInSeq[{{}, {2, x:size(2)}, {}}] = decInSeq[{{}, {1, x:size(2) - 1}, {}}]
    decInSeq[{{}, {1}, {}}]:zero()
    self.parent.decoder:backward(decInSeq, gradLoss)
    self.parent:backwardConnect()

    local zeroTensor = torch.Tensor(x:size(2), x:size(1), Model.cellSizes[#Model.cellSizes]):typeAs(x) -- seqlen x batch
    return self.parent.encoder:backward(x, zeroTensor)
  end
end

return Model
```

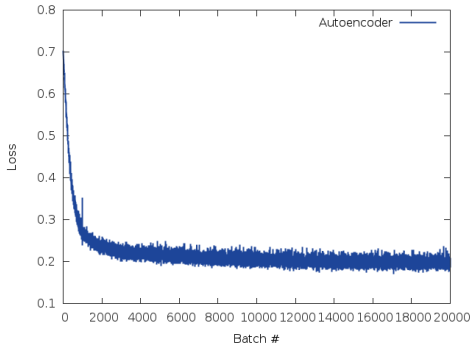
MODEL



TRAINING

Seq2Seq Reconstructions

5.8VAE



VAE TRAINING

```lua
local nn = require 'nn'
require '../modules/Gaussian'

local Model = {
  zSize = 2 -- Size of isotropic multivariate Gaussian Z
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder (inference/recognition model q, variational approximation for posterior p(z|x))
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, 128))
  self.encoder:add(nn.BatchNormalization(128))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(128, 64))
  self.encoder:add(nn.BatchNormalization(64))
  self.encoder:add(nn.ReLU(true))
  -- Create latent Z parameter layer
  local zLayer = nn.ConcatTable()
  zLayer:add(nn.Linear(64, self.zSize)) -- Mean μ of Z
  zLayer:add(nn.Linear(64, self.zSize)) -- Log variance σ^2 of Z (diagonal covariance)
  self.encoder:add(zLayer) -- Add Z parameter layer

  -- Create σε module
  local noiseModule = nn.Sequential()
  local noiseModuleInternal = nn.ConcatTable()
  local stdModule = nn.Sequential()
  stdModule:add(nn.MulConstant(0.5)) -- Compute 1/2 log σ^2 = log σ
  stdModule:add(nn.Exp()) -- Compute σ
  noiseModuleInternal:add(stdModule) -- Standard deviation σ
  noiseModuleInternal:add(nn.Gaussian(0, 1)) -- Sample noise ε ~ N(0, 1)
  noiseModule:add(noiseModuleInternal)
  noiseModule:add(nn.CMulTable()) -- Compute σε


  -- Create sampler q(z) = N(z; μ, σI) = μ + σε (reparametrization trick)
  local sampler = nn.Sequential()
  local samplerInternal = nn.ParallelTable()
  samplerInternal:add(nn.Identity()) -- Pass through μ
  samplerInternal:add(noiseModule) -- Create noise σ * ε
  sampler:add(samplerInternal)
  sampler:add(nn.CAddTable())

  -- Create decoder (generative model q)
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.zSize, 64))
  self.decoder:add(nn.BatchNormalization(64))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(64, 128))
  self.decoder:add(nn.BatchNormalization(128))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(128, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(sampler)
  self.autoencoder:add(self.decoder)
end

return Model

```

VAE MODEL

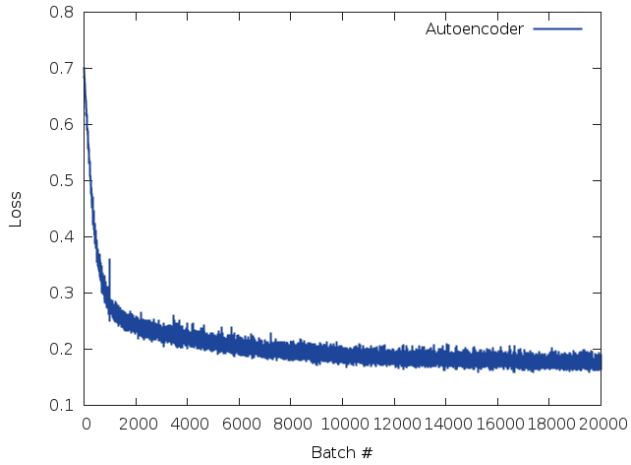
VAE SAMPLES

VAE RECONSTRUCTION

5.9CatVAE

```lua
local nn = require 'nn'
require '../modules/Uniform'

local Model = {
  N = 5, -- Number of Gumbel-(Soft)Max distributions
  k = 10, -- Number of categories/classes
  tau = 1 -- Softmax temperature τ
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder (inference model q, variational approximation for posterior p(z|x))
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, 128))
  self.encoder:add(nn.BatchNormalization(128))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(128, 64))
  self.encoder:add(nn.BatchNormalization(64))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(64, self.N * self.k)) -- Unnormalised log probabilities log(π)

  -- Create noise ε sample module
  local noiseModule = nn.Sequential()
  noiseModule:add(nn.Uniform(0, 1)) -- Sample from U(0, 1)
  -- Transform uniform sample to Gumbel sample
  noiseModule:add(nn.AddConstant(1e-9, true)) -- Improve numerical stability
  noiseModule:add(nn.Log())
  noiseModule:add(nn.MulConstant(-1, true))
  noiseModule:add(nn.AddConstant(1e-9, true)) -- Improve numerical stability
  noiseModule:add(nn.Log())
  noiseModule:add(nn.MulConstant(-1, true))

  -- Create sampler q(z) = G(z) = softmax((log(π) + ε)/τ) (reparametrization trick)
  local sampler = nn.Sequential()
  local samplerInternal = nn.ConcatTable()
  samplerInternal:add(nn.Identity()) -- Unnormalised log probabilities log(π)
  samplerInternal:add(noiseModule) -- Create noise ε
  sampler:add(samplerInternal)
  sampler:add(nn.CAddTable())
  self.temperature = nn.MulConstant(1 / self.tau, true) -- Temperature τ for softmax
  sampler:add(self.temperature)
  sampler:add(nn.View(-1, self.k)) -- Resize to work over k
  sampler:add(nn.SoftMax())
  sampler:add(nn.View(-1, self.N * self.k)) -- Resize back
  -- TODO: Hard sampling

  -- Create decoder (generative model p)
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.N * self.k, 64))
  self.decoder:add(nn.BatchNormalization(64))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(64, 128))
  self.decoder:add(nn.BatchNormalization(128))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(128, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(sampler)
  self.autoencoder:add(self.decoder)
end

return Model
```

CATVAE – Model



CATVAE TRAINING

CATVAE SAMPLES

CATVAERECONSTRUCTIONS

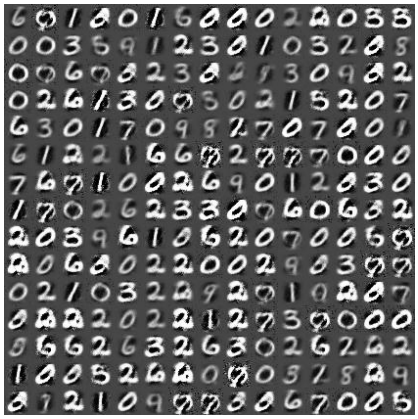
CATVAEINTERPOLATIONS

5.10AAE

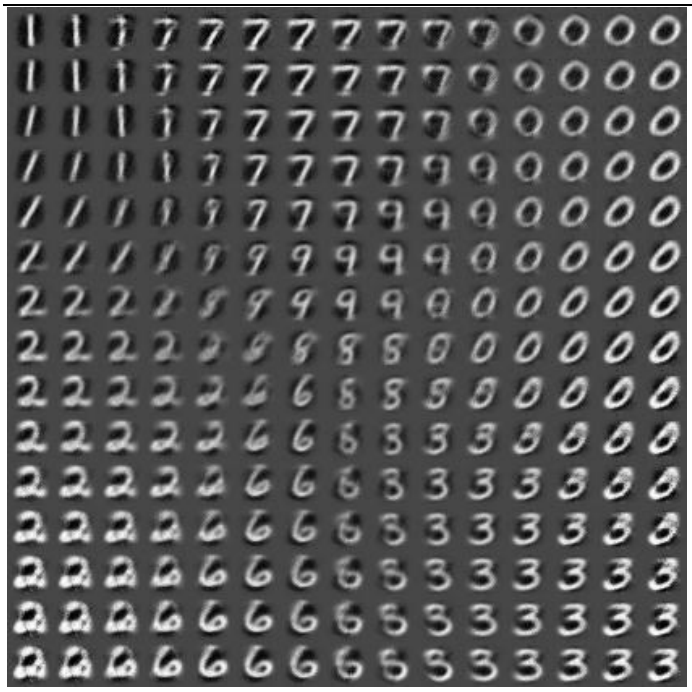
```lua
local nn = require 'nn'

local Model = {
  zSize = 2 --  -- Size of isotropic multivariate Gaussian Z
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder (generator)
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, 128))
  self.encoder:add(nn.BatchNormalization(128))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(128, 64))
  self.encoder:add(nn.BatchNormalization(64))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.Linear(64, self.zSize)) -- Encoding distribution q(z|x) is a deterministic function of x
  -- Note that a Gaussian posterior (like VAE) or universal approximator posterior could be used, but deterministic q(z|x) works well

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.zSize, 64))
  self.decoder:add(nn.BatchNormalization(64))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(64, 128))
  self.decoder:add(nn.BatchNormalization(128))
  self.decoder:add(nn.ReLU(true))
  self.decoder:add(nn.Linear(128, featureSize))
  self.decoder:add(nn.BatchNormalization(featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

function Model:createAdversary()
  -- Create adversary (discriminator)
  self.adversary = nn.Sequential()
  self.adversary:add(nn.Linear(self.zSize, 16))
  self.adversary:add(nn.BatchNormalization(16))
  self.adversary:add(nn.ReLU(true))
  self.adversary:add(nn.Linear(16, 1))
  self.adversary:add(nn.BatchNormalization(1))
  self.adversary:add(nn.Sigmoid(true))
end

return Model

```

AAE MODEL

AAERECONSTRUCTIONS



AAE SAMPLES

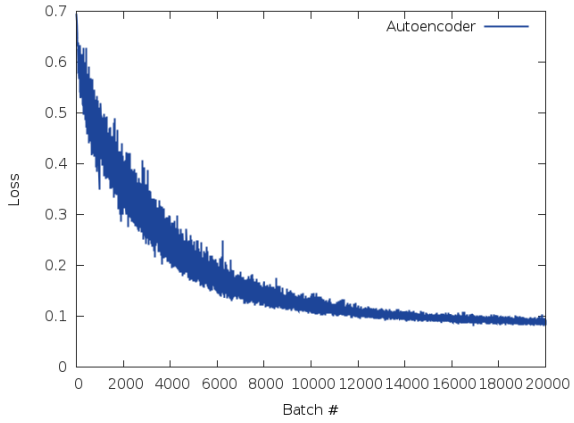


AAE INTERPOLATIONS

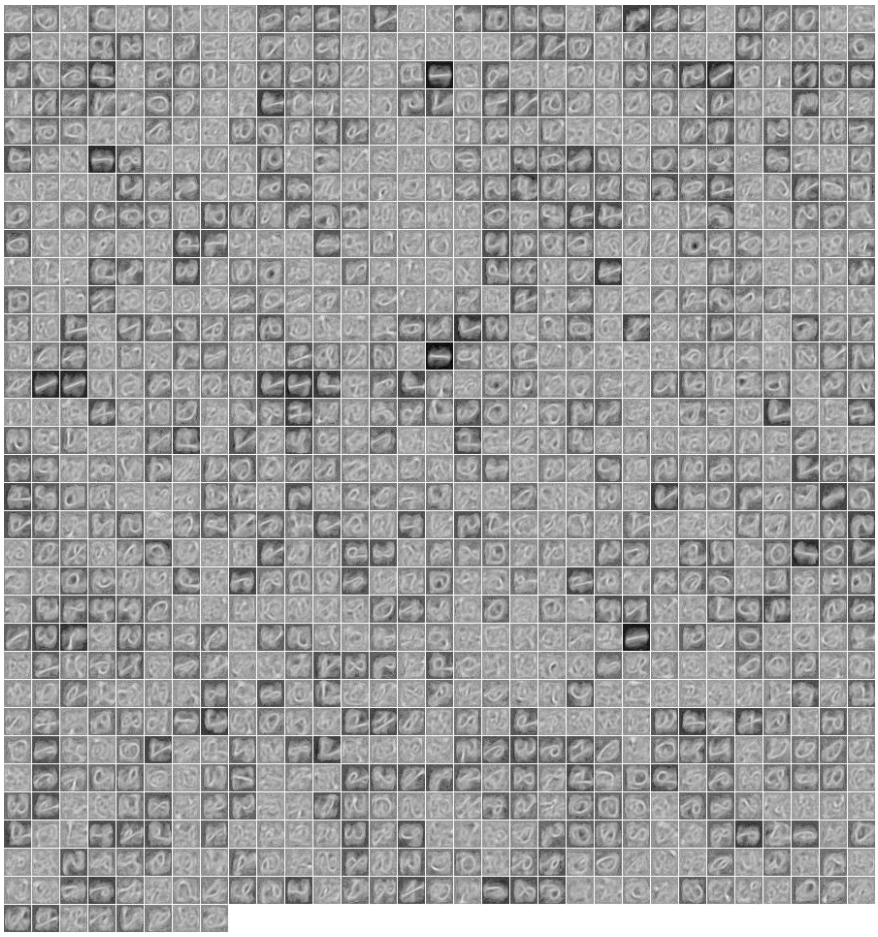
5.11WTA-AE

```lua
local nn = require 'nn'
require '../modules/BatchTopK'

local Model = {
  k = 0.05, -- Sparsity
  features = 1000 -- Number of features
}

function Model:createAutoencoder(X)
  local featureSize = X:size(2) * X:size(3)

  -- Create encoder
  self.encoder = nn.Sequential()
  self.encoder:add(nn.View(-1, featureSize))
  self.encoder:add(nn.Linear(featureSize, self.features))
  self.encoder:add(nn.ReLU(true))
  self.encoder:add(nn.BatchTopK(self.k, true)) -- Extract k% top activations during training

  -- Create decoder
  self.decoder = nn.Sequential()
  self.decoder:add(nn.Linear(self.features, featureSize))
  self.decoder:add(nn.Sigmoid(true))
  self.decoder:add(nn.View(X:size(2), X:size(3)))

  -- Create autoencoder
  self.autoencoder = nn.Sequential()
  self.autoencoder:add(self.encoder)
  self.autoencoder:add(self.decoder)
end

return Model

```

WTA-AE MODEL



WTA-AE TRAINING



WTA-AE WEIGHTS

WTA-AE RECONSTRUCTIONS

Spatial Convolution

Χρονική Εκτέλεση με και χωρίς cuda

Στο συγκεκριμένο παράδειγμα έχουμε εκτελέσει ένα αλγόριθμο εκμάθηση πάνω στα σήματα τροχαίας από την βάση δεδομένων GTSRB, με τον αλγόριθμο Spatial Convolution.

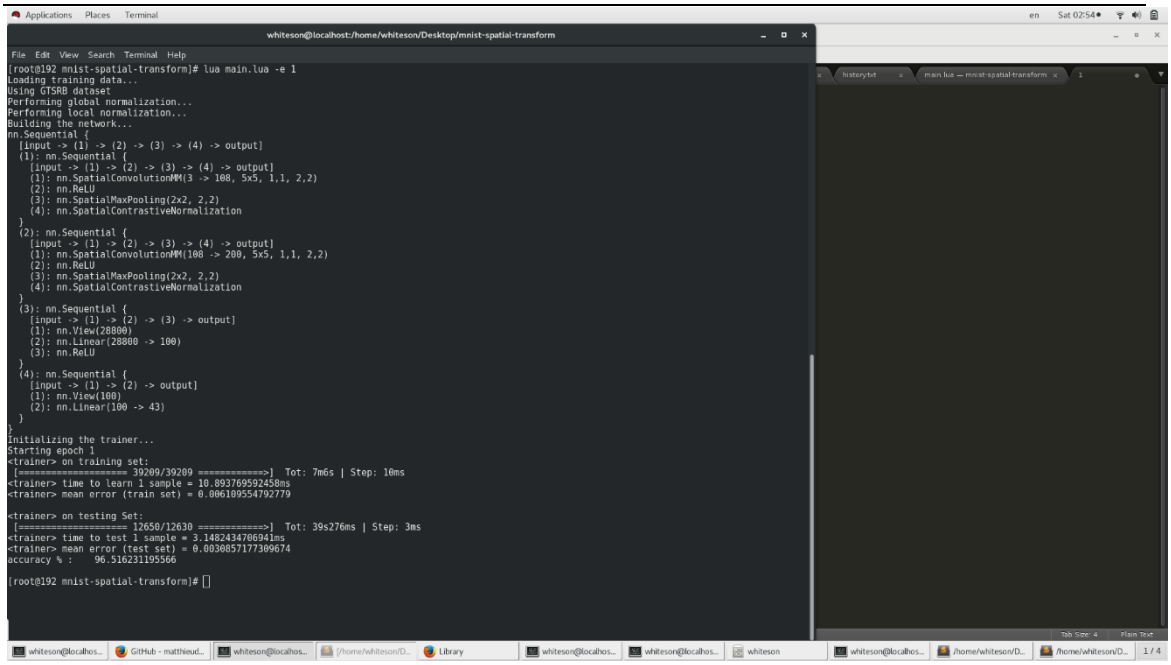
http://benchmark.ini.rub.de/?section=gtsrb&subsection=news

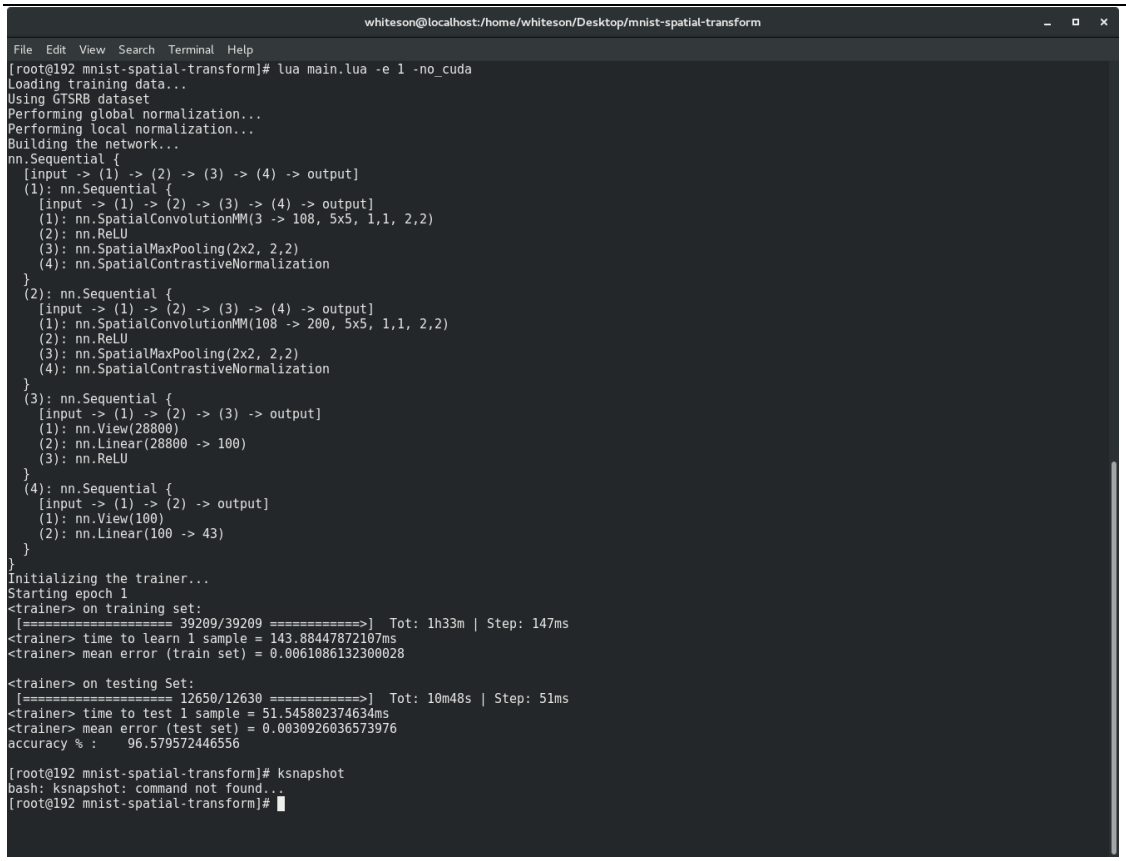
Παρακάτω σε γλώσσα Lua έχουμε εκπαίδευση σε εικόνες τροχαίων σημάτων, όπου ο κώδικας είναι ανοιχτός με πηγή : https://github.com/matthieudelaro/mnist-spatial-transform

εκτελώντας την εντολή:

lua main.lua -e 1,μπορούμε να εκτελέσουμε δυναμικά σε -e=1 epochs με cuda

Παρατηρούμε για ένα dataset των 39.209 εικόνες σημάτων 64x64 Spantial Convolution χρειάστηκαν 7.6λεπτά με κάθε βήμα 10ms και τετραγωνικό λάθος <1% =0.006


Applications Places Terminal
whiteson@localhost:/home/whiteson/Desktop/mnist-spatial-transform
File Edit View Search Terminal Help
[root@192 mnist-spatial-transform]# lua main.lua -e 1
Loading training data...
Using GTSRB dataset
Performing global normalization...
Performing local normalization...
Building the network...
nn.Sequential {
[input -> (1) -> (2) -> (3) -> (4) -> output]
(1): nn.Sequential {
[input -> (1) -> (2) -> (3) -> (4) -> output]
(1): nn.SpatialConvolutionMM(3 -> 108, 5x5, 1,1, 2,2)
(2): nn.ReLU
(3): nn.SpatialMaxPooling(2x2, 2,2)
(4): nn.SpatialContrastiveNormalization
}
(2): nn.Sequential {
[input -> (1) -> (2) -> (3) -> (4) -> output]
(1): nn.SpatialConvolutionMM(108 -> 200, 5x5, 1,1, 2,2)
(2): nn.ReLU
(3): nn.SpatialMaxPooling(2x2, 2,2)
(4): nn.SpatialContrastiveNormalization
}
(3): nn.Sequential {
[input -> (1) -> (2) -> (3) -> output]
(1): nn.View(28800)
(2): nn.Linear(28800 -> 100)
(3): nn.ReLU
}
(4): nn.Sequential {
[input -> (1) -> (2) -> output]
(1): nn.View(100)
(2): nn.Linear(100 -> 43)
}
}
Initializing the trainer...
Starting epoch 1
<trainer> on training set:
[==================== 39209/39209 ============>] Tot: 7m6s | Step: 10ms
<trainer> time to learn 1 sample = 10.893769592458ms
<trainer> mean error (train set) = 0.006109554792779
<trainer> on testing Set:
[==================== 12630/12630 ============>] Tot: 39s276ms | Step: 3ms
<trainer> time to test 1 sample = 3.1482434706941ms
<trainer> mean error (test set) = 0.0030857177309674
accuracy % : 96.516231195566
[root@192 mnist-spatial-transform]#
historytxt
main.lua — mnist-spatial-transform
Tab Size: 4
Plain Text
1 / 4



Ενώ σε εκτέλεση μόνο σε επίπεδο επεξεργαστή έχουμε τα παρακάτω αποτελέσματα:

Παρατηρούμε για ένα dataset των 39.209 εικόνες σημάτων 64x64 Spantial Convolution χρειάστηκαν 1ώρα και 33 λεπτά λεπτά με κάθε βήμα 147ms και τετραγωνικό λάθος <1% =0.006, και testing 10 λεπτά και 51 δευτερόλεπτα.

Επίσης μπορούμε κατά την εκτέλεση να τρέξουμε ένα πρόγραμμα για να λάβουμε την κατάσταση της κάρτας γραφικών που βρίσκεται στο λογισμικό του οδηγού της κάρτας γραφικών μας.

5.12OPENMPI

Παρακάτω ακολουθεί ένα απλό παράδειγμα παράλληλης επεξεργασίας των δεδομένων με το openmpi.

Η προσπέλαση αυτών των δεδομένων, αφού πρώτα έχουν εισαχθεί σε ένα πίνακα, για να μπορέσει να γίνει μια πράξη, πρέπει να γίνει με μία επανάληψη (for, foreach ή while). Σε

περίπτωση που είναι μεγάλα αυτά τα δεδομένα και θελήσουμε να ελαχιστοποιήσουμε το χρόνο διαβάσματος ή επεξεργασίας θα μπορούσαμε να προετοιμάζαμε αυτά για χρήση σε παράλληλο περιβάλλον.

Ένας τρόπος είναι να χωρίσουμε αυτά τα δεδομένα σε μικρότερα αρχεία. Για να μην χάσουμε την σειρά θα πρέπει να ορίσουμε για όνομα ένα αριθμό. Παραδείγματος χάρη τα δεδομένα που είδαμε παραπάνω είναι η τιμή ζήτησης μιας μετοχής και βρίσκονται σε ένα αρχείο prices.txt και έχει 1 εκατομμύριο τιμές. Αυτό που μπορούμε να κάνουμε είναι να χωρίσουμε αυτό το αρχείο σε 100 αρχεία, τα οποία το όνομα τους θα είναι prices1.txt,prices2.txt,prices3.txt... Το πρώτο αρχείο θα περιέχει τις πρώτες 10.000 το prices2.txt τις τιμές 10.001 εώς 20.0000. Με αυτό το τρόπο και με το OpenMpi θα μπορούσαμε να τρέξουμε παράλληλα σε επίπεδο επεξεργαστή η ακόμα και σε cuda, με την παρακάτω εντολή στο τερματικό μας.

openmpi programm.py -n 100

το -n 100 δέιχνει τους πυρήνες που θα απασχολήσουμε αν είχαμε διαθέσιμους, σε περίπτωση που δεν έχουμε θα μπούν στη σειρά εκτέλεσης καθαρά από το λειτουργικό μας σύστημα και το openmpi. Αυτό που θα γίνει είναι να δημιουργηθούν 100 διαδικασίες. Η κάθε διαδικασία φέρει στο πρόγραμμα εκτέλεσης μια μεταβλητή RANK. Αυτή η μεταβλητή είναι που θα χρησιμοποιήσουμε μέσα στο πρόγραμμα για τον ορισμό το αρχείου που εργάζεται η κάθε διαδικασία.

Δηλαδή η εντολή open ('prices1.txt') θα γραφεί ώς open (“prices$RANK.txt”)

Με αυτό το τρόπο η κάθε διαδικασία θα επεξεργαστεί το δικό της αρχείο, και εννοείται ότι μπορεί να έχει ένα οποιοδήποτε αρχείο ώς έξοδο τύπου prices$RANK-out.txt, για κάποια επόμενη διαδικασία ή πρόγραμμα.

6.ΣΥΜΠΕΡΑΣΜΑ

Μετά από αυτή την μελέτη, είδαμε ότι η μηχανική μάθηση μπορεί να υλοποιηθεί με πολύ χαμηλό κόστος υλικών λόγω του ότι υπάρχει η κατάλληλη υποδομή με λογισμικό σε ανοικτό κώδικα, για την υλοποίηση αλγορίθμων μηχανικής μάθησης, παρ ‘όλα αυτά σίγουρα υπάρχουν αρκετές δυσκολίες κατά την υλοποίηση. Κατανοήσαμε ότι πολλά περιβάλλοντα εργασίας χρησιμοποιούν κάποια βασικά εργαλεία, τα οποία δίνουν την δυνατότητα στους ερευνητές να αξιοποιήσουν τους πόρους ενός συστήματος στο έπακρο. Πολλά από αυτά στον πυρήνα τους κρύβουν το openmpi (παραλληλοποίηση σε επίπεδο CPU & CUDA), καθώς και από τους τύπους δεδομένων(Tensor), ο οποίος είναι βασικός τύπος της γλώσσας προγραμματισμού Lua, χρησιμοποιείται με παρόμοιες λειτουργίες στα πιο αποδοτικά frameworks. Η μηχανική μάθηση με παράλληλη επεξεργασία σε κάρτες γραφικών μειώνει σημαντικά το χρόνο εκπαίδευσης με ποσοστιαία μεταβολή τουλάχιστον 12 φορές με μία κάρτα επεξεργασίας γραφικών κοντά στους 600 πυρήνες. Επιλέγουμε για μελλοντική έρευνα την γλώσσα προγραμματισμού Lua λόγω της μεγάλης γκάμας υλοποιημένων αλγορίθμων. Υπάρχει πολύ μεγάλη ομάδα που ασχολείται ενεργά με το framework Torch στη γλώσσα αυτή, το οποίο συνεργάζεται άψογα με κάρτες γραφικής επεξεργασίας. Η βαθειά μηχανική μάθηση σίγουρα χρειάζεται υλικό-λογισμικό με

δυνατότητα παράλληλης επεξεργασίας. Μπορεί τα συνελικτικά δίκτυα να έχουν ανοίξει ένα νέο δρόμο στη μηχανική μάθηση όμως δεν είναι αρκετό για όλα τα προβλήματα. Φαίνεται ότι στο μέλλον θα υπάρξουν αρκετές εξελίξεις στον χώρο της μηχανικής μάθησης και κατ' επέκταση στην τεχνητή νοημοσύνη. Καταλαβαίνουμε επίσης ότι η βαθειά μάθηση θα είναι ένα εργαλείο που θα μπορέσει με συνεργασία άλλων να παράγει τεχνητή νοημοσύνη.

7.ΜΕΛΛΟΝΤΙΚΗ ΕΡΓΑΣΙΑ

Μελλοντική εργασία θα είναι η έρευνα στους αλγόριθμους εκμάθησης, και στους τρόπους υλοποίησης με στόχο την γρηγορότερη εκπαίδευση υπολογιστικών συστημάτων, καθώς και ανάπτυξη μηχανισμών ώστε η μηχανική μάθηση να μπορέσει να λύσει περισσότερα προβλήματα, πιο αποδοτικά και πιο αξιόπιστα. Συγκεκριμένα αυτό που θα πειραματιστούμε είναι η μηχανική μάθηση σε απεικονίσεις όπου η είσοδος, θα είναι ήδη σε συμπιεσμένη μορφή ,και αυτό θα έχει σαν αποτέλεσμα την γρηγορότερη εκμάθηση, αρκεί να υπάρχει ο σωστός μηχανισμός που θα δημιουργεί ένα προς ένα μια απεικόνιση της εισόδου ως αποτύπωμα. Η συμπίεση των δεδομένων θα είναι το πρώτο επίπεδο επεξεργασίας της και στην συνέχεια οι βαθειάς μάθησης αλγόριθμοι θα αναλάβουν την μάθηση.

8. ΒΙΒΛΙΟΓΡΑΦΙΑ

http://learningsys.org/papers/LearningSys_2015_paper_33.pdf

https://arxiv.org/pdf/1409.1556.pdf

https://arxiv.org/pdf/1512.00567.pdf

https://arxiv.org/pdf/1512.03385.pdf

https://arxiv.org/pdf/1611.05431.pdf

https://pjreddie.com/media/files/papers/yolo.pdf

https://arxiv.org/pdf/1511.00561.pdf

https://developer.ibm.com/articles/cc-machine-learning-deep-learning-architectures/

Deep Learning,by Adam Gibson, Josh Patterson,Publisher: O'Reilly Media, Inc.,Release Date: August 2017,ISBN: 9781491924570

https://project.inria.fr/deeplearning/files/2016/05/DLFrameworks.pdf

https://dzone.com/articles/8-best-deep-learning-frameworks

https://csce.ucmss.com/cr/books/2017/LFS/CSREA2017/ABD3177.pdf

https://arxiv.org/pdf/1605.02688.pdf

https://www.lua.org

www.hdfgroup.org

https://infoscience.epfl.ch/record/192376/files/Collobert_NIPSWORKSHOP_2011.pdf

https://core.ac.uk/download/pdf/81214805.pdf

Léonard, Nicholas, Sagar Waghmare, Yang Wang, and Jin-Hwa Kim. rnn: Recurrent Library for Torch. arXiv preprint arXiv:1511.07889 (2015).

https://www.marutitech.com/top-8-deep-learning-frameworks/

https://github.com/Kaixhin/Autoencoders

https://developer.nvidia.com/deep-learning-frameworks

https://dzone.com/articles/8-best-deep-learning-frameworks

http://learningsys.org/papers/LearningSys_2015_paper_33.pdf

http://cs242.stanford.edu/assets/projects/2017/liubaige-xzang.pdf

https://csce.ucmss.com/cr/books/2017/LFS/CSREA2017/ABD3177.pdf

https://singa.incubator.apache.org/en/docs/index.html

https://www.tensorflow.org/deploy/distributed

https://dl.acm.org/citation.cfm?id=2807410

http://deeplearning.net/software/theano/theano.pdf

https://www.uio.no/studier/emner/matnat/ifi/INF5860/v17/timeplan/architectures-in-deep-learning.pdf