ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Βαθιάς Μηχανικής Μάθησης και Εξομάλυνση με χρήση της μεθόδου Dropout

Χαρίλαος Σ. Παπαϊωάννου

Επιβλέπων: **Σέργιος Θεοδωρίδης,** Καθηγητής

ΑΘΗΝΑ

Μάρτιος 2018

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Αλγόριθμοι Βαθιάς Μηχανικής Μάθησης και Εξομάλυνση με χρήση της μεθόδου Dropout

Χαρίλαος Σ. Παπαϊωάννου

Α.Μ.: Μ 1447

ΕΠΙΒΛΕΠΩΝ: **Σέργιος Θεοδωρίδης,** Καθηγητής

ΕΞΕΤΑΣΤΙΚΗ ΕΠΙΤΡΟΠΗ: **Σέργιος Θεοδωρίδης,** Καθηγητής
Δημήτριος Βαρουτάς, Αναπληρωτής Καθηγητής

Μάρτιος 2018

ΠΕΡΙΛΗΨΗ

Σκοπός της παρούσας εργασίας είναι η ανάπτυξη μιας πληθώρας αλγορίθμων βαθιάς Μηχανικής Μάθησης και η έρευνα της σημασίας που έχει η εξομάλυνση με Dropout σε αυτούς. Η μεθοδολογία που ακολουθείται, αποτελείται από την μελέτη όλων των βασικών σημείων της θεωρίας κάθε μεθόδου, την προγραμματιστική ανάπτυξή της και, τέλος, τη διενέργεια πειραμάτων για τη διερεύνηση των επιμέρους θεμάτων που μας ενδιαφέρουν.

Το σύνολο της εργασίας κινείται στο ευρύτερο πεδίο των Τεχνητών Νευρωνικών δικτύων. Για τον λόγο αυτό, θεωρήθηκε σκόπιμη μια αναλυτική παρουσίαση των θεωρητικών τους βάσεων προτού περάσουμε στη μελέτη των πιο σύγχρονων τεχνικών. Τα πρώτα κεφάλαια εξυπηρετούν ακριβώς αυτόν τον σκοπό.

Στο τρίτο κεφάλαιο παρουσιάζεται η θεωρία γύρω από την εξομάλυνση καθώς και αναλυτικά η μέθοδος Dropout. Στα επόμενα κεφάλαια, μέχρι τα Συμπεράσματα, ακολουθείται η προσέγγιση που αναφέρθηκε παραπάνω. Μετά από τη θεωρητική παρουσίαση κάθε αλγορίθμου, προχωράμε στην ανάπτυξή του με χρήση της βιβλιοθήκης TensorFlow και, τελικά, εξετάζουμε τα πειραματικά μας αποτελέσματα.

Βλέπουμε πολλές αρχιτεκτονικές βαθιών πολυστρωματικών Νευρωνικών δικτύων για την αναγνώριση ψηφίων γραμμένων με το χέρι καθώς και ένα Convolutional Neural Network για τον ίδιο σκοπό. Ακόμα, μελετάμε την υλοποίηση ενός Recurrent Neural Network για τη μοντελοποίηση γλώσσας.

Σε όλους τους αλγορίθμους αναζητήσαμε τη σημασία της εξομάλυνσης μέσω του Dropout και διαπιστώσαμε πειραματικά την αύξηση της απόδοσης όλων, τη βελτίωση της γενικευτικής ικανότητας κάθε μοντέλου, όταν σε αυτό εφαρμόσαμε τη συγκεκριμένη μέθοδο εξομάλυνσης.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Μάθηση

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: βαθιά-μηχανική-μάθηση, νευρωνικά-δίκτυα, εξομάλυνση, dropout, tensorflow

ABSTRACT

The goal of this thesis is the study of a variety of deep learning algorithms and the evaluation of their regularization with Dropout technique. The methodology followed, begins with the presentation of the key theoretical points for each algorithm, followed by a software program for each one of them and concludes with the presentation of the experimental results for the aspects we are interested in.

The whole thesis lies in the scientific field of Artificial Neural Networks. This is the reason we come across an analytical presentation of the main theory of Neural Networks in the first chapters, before we move forward to the modern techniques.

In the third chapter, the regularization theory is presented along with a detailed presentation of the Dropout method. In the next chapters, until the thesis' conclusion, the methodology stated above is followed. After the theoretical analysis for each algorithm, its development is implemented with the usage of a deep learning library, TensorFlow, and in the end the experimental results get investigated.

We see various different architectures of fully-connected Feed Forward Neural Networks for the handwritten digits recognition problem as well as a Convolutional Neural Network for the same task. We, also, study a Recurrent Neural Network for language modeling.

For all algorithms used, the practical significance of their regularization with Dropout has been investigated, and we ascertained experimentally that it boosted their performance and it improved the ability for each model to generalize.

SUBJECT AREA: Machine Learning

KEYWORDS: deep-learning, neural-networks, regularization, dropout, tensorflow

Στον Σέργιο και σε όλους τους δασκάλους μου

είτε εξασκούσαν αυτό το επάγγελμα είτε όχι

ΕΥΧΑΡΙΣΤΙΕΣ

Φτάνοντας στο τελείωμα άλλης μια εξαιρετικά ενδιαφέρουσας διαδρομής, νιώθω ότι θέλω να ευχαριστήσω τους κοντινούς μου ανθρώπους. Από τη θέση του ο καθένας με βοήθησε, και συνεχίζει να με βοηθάει, στο να έχω καθαρότερη σκέψη. Σας ευχαριστώ.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΣΧΗΜΑΤΩΝ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

ΚΑΤΑΛΟΓΟΣ ΠΙΝΑΚΩΝ

ΠΡΟΛΟΓΟΣ

Η παρούσα εργασία διενεργήθηκε στο πλαίσιο του Προγράμματος Μεταπτυχιακών Σπουδών του Τμήματος Πληροφορικής και Τηλεπικοινωνιών. Το ευρύτερο γνωστικό πεδίο στο οποίο ανήκει είναι εκείνο την Μηχανικής Μάθησης. Η φιλοσοφία που ακολουθήθηκε για την εκπόνησή της ανάγεται στην συμπαγή παρουσίαση της θεωρίας κάθε θέματος, την ανάπτυξή του στη συνέχεια με προγραμματιστικά εργαλεία και τέλος στην έρευνα επιμέρους μεθόδων που μπορούν να χρησιμοποιηθούν για κάποιον σκοπό σε αυτό.

Η διάρθρωση της εργασίας είναι σε κεφάλαια και παραγράφους, σπανιότερα σε υποπαραγράφους. Μετά από μια ολιγοσέλιδη εισαγωγή στο ευρύτερο επιστημονικό πεδίο που ανήκει η εργασία, ο αναγνώστης θα συναντήσει μια αναλυτική παρουσίαση όλων των βασικών εννοιών των νευρωνικών δικτύων μέσα από μια χρονική μετατόπιση στην ιστορία τους, από τις ιδέες που τα διαμόρφωσαν έως το πιο πρόσφατο deep learning.

Εδώ, θα ήθελα να κάνω μια σημείωση. Στο κείμενο χρησιμοποιείται συχνά ορολογία μόνο στα αγγλικά ή κάποιες φορές ο ίδιος όρος σε ελληνικά και αγγλικά. Αυτή η χρήση προέκυψε κατ' επιλογή, με βάση την εμπειρική χρήση των όρων αυτών και με κριτήριο την πληρέστερη πληροφόρηση του αναγνώστη.

Επιστρέφοντας στη δομή της εργασίας, στο τρίτο κεφάλαιο παρουσιάζεται η έννοια της εξομάλυνσης, οι λόγοι για τους οποίους είναι χρήσιμη έως αναγκαία η εφαρμογή της και οι βασικές μέθοδοι με τις οποίες επιτυγχάνεται. Ιδιαίτερη αναφορά γίνεται στη μέθοδο Dropout που διαπερνά σαν συνεχή γραμμή όλα τα επόμενα κεφάλαια και την έρευνα που γίνεται σε αυτά.

Το τέταρτο κεφάλαιο είναι αρκετά σημαντικό καθώς στο πρώτο μέρος του υπάρχει μια αναλυτική εισαγωγή στο TensorFlow, την προγραμματιστική βιβλιοθήκη που παρέχει τα εργαλεία για την ανάπτυξη αλγορίθμων βαθιάς μηχανικής μάθησης, η οποία χρησιμοποιείται στην εργασία. Στη συνέχεια, παρουσιάζεται η ανάπτυξη ενός πολυστρωματικού νευρωνικού δικτύου. Τέλος, λαμβάνουμε τα πειραματικά αποτελέσματα της απόδοσης διάφορων αρχιτεκτονικών στο πρόβλημα της αναγνώρισης ψηφίων γραμμένων με το χέρι.

Όπως το τέταρτο κεφάλαιο, έτσι και τα επόμενα δύο περιέχουν την αντίστοιχη θεωρία, την προγραμματιστική ανάπτυξη και τέλος τα αντίστοιχα πειραματικά αποτελέσματα. Στο πέμπτο κεφάλαιο, αφού παρουσιαστεί η θεωρία των Convolutional Neural Networks, αναλύεται η υλοποίησή τους στο TensorFlow και εξετάζονται τα αποτελέσματα των πειραμάτων για το ίδιο με το προηγούμενο κεφάλαιο πρόβλημα.

Στο επόμενο, έκτο, κεφάλαιο παρουσιάζονται τα Recurrent Neural Networks καθώς και μια συγκεκριμένη οικογένεια αυτών, τα Long Short-Term Memory δίκτυα. Μελετάται, ακόμα, η ανάπτυξη ενός τέτοιου νευρωνικού δικτύου για το πρόβλημα μοντελοποίησης γλώσσας και εξετάζεται πειραματικά η σημασία της εξομάλυνσης με Dropout σε αυτό.

Τέλος, στο έβδομο κεφάλαιο σημειώνονται τα πιο σημαντικά θέματα της εργασίας και συνοψίζονται τα συμπεράσματα που προέκυψαν από τη μελέτη που έγινε σε αυτή.

1. ΕΙΣΑΓΩΓΗ

Η Επιστήμη είναι ένας τρόπος σκέψης μέσω του οποίου επιδιώκουμε αυτό που ονομάζουμε γνώση. Γνώση ονομάζουμε την ακριβή κατανόηση των πραγμάτων, τόσο της κατάστασης τους όσο και των διαδικασιών από τις οποίες προήλθαν και εκείνων που πιθανόν να μετάσχουν. Πώς θα προκύψει η γνώση της πραγματικότητας; Μέσω της επιστημονικής μεθόδου, της διατύπωσης μιας θεωρίας και της επαλήθευσής της ή όχι μέσω του πειραματισμού.

Στην περίπτωση που η θεωρία μας επαληθεύεται, αυτομάτως σημαίνει ότι κατακτήσαμε την γνώση της πραγματικότητας; Δυστυχώς, όχι. Τα παραδείγματα πολλά, όπως η επέκταση της Νευτώνειας μηχανικής από τη θεωρία της σχετικότητας του Αϊνστάιν. Αν είναι, όμως, έτσι τα πράγματα, μπορούμε να καυχιόμαστε για την καθολική επικράτηση της Επιστήμης; Η απάντηση είναι και εδώ αρνητική. Αλλά, δεν παύει να ισχύει το γεγονός ότι η επιστημονική μέθοδος είναι ό,τι καλύτερο έχουμε στη διάθεσή μας, στον δρόμο για την βαθύτερη κατανόηση του κόσμου που μας περιβάλλει.

Έχοντας τα παραπάνω στην άκρη του μυαλού μας, ίσως μπορούμε να δούμε τις διατυπωμένες επιστημονικές θεωρίες υπό ένα περισσότερο διαλλακτικό βλέμμα. Ίσως μπορούμε να αναρωτηθούμε για τον βαθμό της προσέγγισης που αυτές περικλείουν. Μπορούμε, τότε, με λίγη περισσότερη τόλμη να τις αντιμετωπίσουμε σαν μοντέλα που προσπαθούν να περιγράψουν με όσο πιο ακριβή τρόπο την πραγματικότητα.

Οι λόγοι που επιθυμούμε την κατάκτηση της γνώσης είναι αρκετοί και μπορεί να ποικίλλουν από άνθρωπο σε άνθρωπο. Μπορεί να είναι η διάθεση για εξερεύνηση. Μπορεί η ανάγκη για απαντήσεις, η πρακτική εφαρμογή ίσως της κατανόησης αυτής για κάποιο σκοπό. Λαμβάνοντας την, πιο προσιτή, τελευταία περίπτωση γνωρίζουμε ότι αυτή ανάγεται στην επιθυμία του ανθρώπου να κάνει προβλέψεις ή να είναι σε θέση να κατανοήσει υπό διαφορετικά πρίσματα το παρόν.

Το εκπληκτικό είναι ότι υπάρχουν πάρα πολλές καταστάσεις στις οποίες η επιθυμία αυτή είναι δυνατό να μετουσιωθεί. Γνωρίζοντας όλες τις παραμέτρους, είμαστε σε θέση να προβλέψουμε την τροχιά μια μπάλας αφού αυτή φύγει από τα χέρια κάποιου αθλητή. Βέβαια, υπάρχουν προβλήματα στα οποία δεν έχουμε την αναλυτική περιγραφή τους, προβλήματα που αποδεδειγμένα δεν είναι εφικτό να υπολογίσουμε τη λύση τους σε πεπερασμένο χρόνο [1]. Ένα παράδειγμα, που ίσως μας ενδιαφέρει αρκετά, είναι η αδυναμία υπολογισμού της κατάστασης ισορροπίας ενός οικονομικού συστήματος [2].

Ακόμα, όμως, και να μην έχουμε τη δυνατότητα υπολογισμού της ορθής λύσης, επιθυμούμε να αναπτύξουμε το καλύτερο δυνατό μοντέλο για την προσέγγιση αυτής. Η διαδικασία αυτή είναι, κατά κανόνα, υπολογιστικά ακριβή. Εδώ είναι που, η παράλληλη πορεία της συνεχούς βελτίωσης στην τεχνολογία των υπολογιστών έρχεται να συμβάλλει στη διαδικασία ανάπτυξης κατάλληλων μοντέλων.

Μια πληθώρα αλγορίθμων έχουν διατυπωθεί που περιγράφουν τον τρόπο με τον οποίο θα οδηγηθούμε στη διαμόρφωση μοντέλων με τη χρήση των υπολογιστών. Όλες αυτές οι μέθοδοι περιλαμβάνονται στον όρο Μηχανική Μάθηση. Διαμορφώνουμε κατάλληλες μεθόδους που εκμεταλλεύονται την υπολογιστική ικανότητα των μηχανών ώστε να καταλήξουμε σε όσο το δυνατόν καλύτερες λύσεις, σε προβλήματα που δεν έχουμε τη δυνατότητα να επιλύσουμε αναλυτικά.

Σκοπός των αλγορίθμων Μηχανικής Μάθησης είναι το επιτυχημένο ταίριασμά τους σε ένα περιβάλλον δεδομένων, η στατιστική κατανόησή του συνόλου των δεδομένων, με άλλα λόγια. Με αυτόν τον τρόπο, μπορούμε να αναπτύξουμε ένα αποτελεσματικό μοντέλο που θα βοηθάει τον γιατρό στη διάγνωση της ύπαρξης ή όχι μιας πάθησης σε ένα ασθενή. Μπορούμε σίγουρα να υποβοηθήσουμε τον οδηγό με μοντέλα που έχουν

την ικανότητα να προβλέψουν και να αποσοβήσουν κάποιο ατύχημα. Αυτές είναι δύο πολύ σημαντικές κατηγορίες της Μηχανικής Μάθησης. Η ταξινόμηση των δεδομένων στις σωστές κατηγορίες και η πρόβλεψη της επόμενης τιμής με δεδομένες τις προηγούμενες.

Σε πολλές περιπτώσεις, οι αλγόριθμοι που αναπτύσσουμε έχουν ως σημείο εκκίνησης μια παρατήρησή μας από τη φύση. Αυτό είναι απόλυτα εμφανές στην οικογένεια μεθόδων την οποία μελετά η παρούσα εργασία, τα νευρωνικά δίκτυα που από την ονομασία τους, ήδη, καταλαβαίνουμε ότι θα επιχειρήσουν να προσομοιάσουν την διαδικασία μάθησης του ανθρώπου.

2. ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΚΑΙ DEEP LEARNING

2.1 Ιστορική Αναδρομή

Τα Τεχνητά Νευρωνικά Δίκτυα (Artificial Neural Networks) έχουν μια μακρά ιστορία που ξεκινάει από τις πρώτες προσπάθειες να κατανοήσουμε τον τρόπο με τον οποίο λειτουργεί ο ανθρώπινος (και γενικότερα ο ζωικός) εγκέφαλος καθώς και πως αυτό που ονομάζουμε νοημοσύνη δομείται [3, σσ 875–933].

Το δομικό στοιχείο του εγκεφάλου είναι ο νευρώνας (neuron). Ο ανθρώπινος εγκέφαλος αποτελείται από περίπου 60 με 100 δισεκατομμύρια νευρώνες, ένας αριθμός της τάξης του πλήθους των άστρων στον Γαλαξία μας. Κάθε νευρώνας συνδέεται με άλλους νευρώνες μέσω συνδέσεων που ονομάζονται συνάψεις (synapses). Εκτιμάται ότι υπάρχουν περίπου 50 με 100 τρισεκατομμύρια συνάψεις στον ανθρώπινο εγκέφαλο. Ο πιο συνηθισμένος τύπος συνάψεων είναι οι χημικές, οι οποίες μετατρέπουν ηλεκτρικούς παλμούς, παραγόμενους από κάποιο νευρώνα, σε χημικά σήματα και στη συνέχεια πάλι σε ηλεκτρικά. Ανάλογα τον παλμό (ή τους παλμούς) εισόδου, μια σύναψη είναι αντίστοιχα ενεργή ή όχι. Μέσω των συνδέσεων αυτών, κάθε νευρώνας είναι συνδεδεμένος με άλλους νευρώνες σε μια ιεραρχική δομή που αποτελείται από διαφορετικά στρώματα.

Η ιστορία των (τεχνητών) νευρωνικών δικτύων ξεκινάει το 1943 με την πρωτοποριακή εργασία των McCulloch και Pitts, [4], με την ανάπτυξη ενός υπολογιστικού μοντέλου για τον βασικό νευρώνα. Επιπλέον, οι δύο επιστήμονες παρείχαν αποτελέσματα τα οποία ενώνουν τη νευρολογία με τα μαθηματικά. Συγκεκριμένα, έδειξαν ότι δοθέντος ενός ικανού αριθμού νευρώνων και ρυθμίζοντας κατάλληλα τις μεταξύ τους συνδέσεις, όπου κάθε μία αναπαριστάται από ένα βάρος (weight), μπορούμε - θεωρητικά - να υπολογίσουμε οποιαδήποτε υπολογίσιμη συνάρτηση.

Στη συνέχεια, ο Frank Rosenblatt, [5], χρησιμοποιώντας το μοντέλο του νευρώνα των McCulloch και Pitts, ανέπτυξε μια πραγματική μηχανή ικανή να μάθει (learning machine) από ένα σύνολο δεδομένων εκμάθησης (training data). Στην απλούστερη μορφή της μηχανής, χρησιμοποίησε έναν μοναδικό νευρώνα και εφάρμοσε έναν κανόνα ώστε να μπορεί να διαχωρίζει τα δεδομένα (data) που άνηκαν σε δύο γραμμικώς διαχωρίσιμες κλάσεις. Έφτιαξε, δηλαδή, ένα σύστημα αναγνώρισης προτύπων. Ονόμασε τη μηχανή αυτή perceptron και ανέπτυξε έναν αλγόριθμο για την εκπαίδευσή του.

Τα νευρωνικά δίκτυα είναι, πλέον, μηχανές μάθησης αποτελούμενες από μεγάλο αριθμό νευρώνων συνδεδεμένους μεταξύ τους σε στρώματα (layers). Η εκμάθηση επιτυγχάνεται ρυθμίζοντας τα άγνωστα συναπτικά βάρη με σκοπό την ελαχιστοποίηση μιας προεπιλεγμένης συνάρτησης κόστους. Χρειάστηκαν περισσότερα από 25 χρόνια από την πρωτοποριακή δουλειά του Rosenblatt, για να βρουν τα νευρωνικά δίκτυα την ευρεία χρήση τους στη μηχανική μάθηση (machine learning). Κομβικό σημείο ήταν η ανάπτυξη του αλγορίθμου backpropagation (οπισθοδρόμησης σφάλματος) για την εκμάθηση νευρωνικών δικτύων βάσει ενός συνόλου εκπαίδευσης που περιέχει ζεύγη εισόδου-εξόδου.

Ενδιαφέρον παρουσιάζει το γεγονός ότι τα νευρωνικά δίκτυα κυριαρχούσαν στο πεδίο της μηχανικής μάθησης για μια δεκαετία, από το 1986 μέχρι τα μέσα της δεκαετίας του 1990. Στη συνέχεια, παραγκωνίστηκαν και τα support vector machines (μηχανές υποστήριξης διανυσμάτων) πήραν κατά κύριο λόγο τη θέση τους. Τα τελευταία χρόνια, υπάρχει μια ραγδαία αύξηση του ενδιαφέροντος για τα νευρωνικά δίκτυα στο πλαίσιο του deep learning (βαθιά μηχανική μάθηση). Πολλά layers επιτρέπουν τις κατάλληλες, μετά από εκπαίδευση, μεταβολές στην αναπαράσταση των δεδομένων (data

representation) για την ελαχιστοποίηση του σφάλματος της εξόδου συγκρινόμενη με την επιθυμητή.

2.2 Το Perceptron

Πρωταρχικό σημείο για τα νευρωνικά δίκτυα είναι η ανάπτυξη του Perceptron το οποίο επιλύει το απλό πρόβλημα κατηγοριοποίησης των δεδομένων (classification) σε δύο γραμμικά διαχωρίσιμες κλάσεις (ω_1, ω_2). [3, σ 878]

Με άλλα λόγια, δίνουμε ένα σύνολο από δείγματα εκπαίδευσης $(y_n, \boldsymbol{x}_n)$, $n = 1, 2, \ldots, N$, με $y_n \in \{-1, +1\}$, $\boldsymbol{x}_n \in R^l$ και θεωρούμε ότι υπάρχει ένα υπερεπίπεδο (hyperplane),

$$\boldsymbol{\theta}_*^T \boldsymbol{x} = 0$$

τέτοιο ώστε,

$$\boldsymbol{\theta}_*^T \boldsymbol{x} > 0, \text{ αν } \boldsymbol{x} \in \omega_1,$$

$$\boldsymbol{\theta}_*^T \boldsymbol{x} < 0, \text{ αν } \boldsymbol{x} \in \omega_2.$$

Το υπερεπίπεδο αυτό διαχωρίζει πλήρως τις 2 κλάσεις, ταξινομώντας σωστά όλα τα δείγματα εκπαίδευσης. Σημειώνεται ότι για απλότητα συμβολισμού, ο bias όρος έχει ενσωματωθεί στο $\boldsymbol{\theta}_*$ με επέκταση της διάστασης του προβλήματος κατά μία, δηλαδή όπου $\boldsymbol{x}$ θεωρούμε $[\boldsymbol{x}\ 1]^T$.

Ο στόχος είναι η ανάπτυξη ενός αλγορίθμου ο οποίος επαναληπτικά υπολογίζει ένα υπερεπίπεδο που κατηγοριοποιεί σωστά όλα τα πρότυπα (δείγματα εκπαίδευσης) και από τις δύο κλάσεις. Για τον σκοπό αυτό, μια συνάρτηση κόστους υιοθετείται.

2.2.1 Η συνάρτηση κόστους του Perceptron

Έστω η διαθέσιμη εκτιμήτρια (estimate) στο τρέχον επαναληπτικό βήμα είναι $\boldsymbol{\theta}$. Τότε, υπάρχουν δύο περιπτώσεις. Η πρώτη είναι όλα τα σημεία (πρότυπα) να έχουν κατηγοριοποιηθεί σωστά κάτι που σημαίνει ότι μια λύση έχει βρεθεί. Η άλλη είναι ότι η εκτιμήτρια $\boldsymbol{\theta}$ κατηγοριοποιεί σωστά κάποια από τα πρότυπα ενώ τα υπόλοιπα όχι. Έστω $\boldsymbol{Y}$ το σύνολο των σημείων που έχουν κατηγοριοποιηθεί λάθος. Η συνάρτηση κόστους του Perceptron ορίζεται ως:

$$\boldsymbol{J}(\boldsymbol{\theta}) = -\sum_{n:\boldsymbol{x}_n \in Y} y_n \boldsymbol{\theta}^T \boldsymbol{x}_n$$

όπου,

$$y_n = \begin{cases} +1, \alpha\nu\ \boldsymbol{x} \in \omega_{\boldsymbol{1}} \\ -1, \alpha\nu\ \boldsymbol{x} \in \omega_{\boldsymbol{2}} \end{cases}$$

Παρατηρούμε ότι η συνάρτηση κόστους είναι μη αρνητική. Συγκεκριμένα, λόγω του αθροίσματος που είναι στο σύνολο των λανθασμένα κατηγοριοποιημένων σημείων, $\alpha\nu\ \boldsymbol{x_n} \in \omega_1\ (\omega_2)\ \tau\acute{o}\tau\varepsilon\ \boldsymbol{\theta}^T \boldsymbol{x}_n < (>) 0$ οδηγώντας το γινόμενο $-y_n \boldsymbol{\theta}^T \boldsymbol{x}_n > 0$. Η συνάρτηση κόστους γίνεται μηδέν, στην περίπτωση που δεν υπάρχουν πρότυπα ταξινομημένα σε λάθος κλάση, δηλαδή όταν $Y = \emptyset$, κάτι που αντιστοιχεί σε λύση.

Γράφοντας τη συνάρτηση κόστους με ελαφρώς διαφοροποιημένο τρόπο,

$$\boldsymbol{J}(\boldsymbol{\theta}) = \left(- \sum_{n:\boldsymbol{x}_n \in Y} y_n \boldsymbol{x}_n^T \right) \boldsymbol{\theta}$$

παρατηρούμε ότι αυτή είναι γραμμική συνάρτηση του $\boldsymbol{\theta}$, όσο το πλήθος των άστοχων κατηγοριοποιήσεων παραμένει ίδιο. Η μεταβολή στην τιμή του $\boldsymbol{\theta}$ αντιστοιχεί σε αλλαγή στη θέση του διαχωριστικού υπερεπιπέδου και συνεπώς σε μεταβολή του πλήθους των άστοχων κατηγοριοποιήσεων.

2.2.2 Ο αλγόριθμος του Perceptron

Αποδεικνύεται, [6], ότι ξεκινώντας από ένα αυθαίρετο σημείο, $\boldsymbol{\theta}^{(0)}$, η ακόλουθη επαναληπτική ενημέρωση,

$$\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu_i \sum_{n:\boldsymbol{x}_n \in Y} y_n \boldsymbol{x}_n^T$$

συγκλίνει έπειτα από πεπερασμένο αριθμό επαναλήψεων. Η παράμετρος μ_i επιλέγεται κατάλληλα ώστε να εγγυάται τη σύγκλιση.

Μια άλλη μορφή του αλγορίθμου δέχεται ένα πρότυπο κάθε φορά, σε μια κυκλική προσέγγιση, έως ότου να υπάρξει σύγκλιση. Συμβολίζοντας με $y_{(i)}$, $\boldsymbol{x}_{(i)}$, $(i) \in \{1, 2, \ldots, N\}$, το εκπαιδευτικό ζεύγος που παρουσιάζεται στον αλγόριθμο στο $i - οστό$ βήμα, η επαναληπτική ενημέρωση γίνεται

$$\boldsymbol{\theta}^{(i)} = \begin{cases} \boldsymbol{\theta}^{(i-1)} + \mu_i y_{(i)} \boldsymbol{x}_{(i)}, & αν\ το\ \boldsymbol{x}_{(i)}\ ταξινομήθηκε\ λάθος\ από\ τη\ \boldsymbol{\theta}^{(i-1)} \\ \boldsymbol{\theta}^{(i-1)}, & αλλιώς \end{cases}$$

Εκκινώντας από μία αρχική εκτίμηση, συνήθως $\boldsymbol{\theta}^{(0)} = 0$, ελέγχουμε κάθε ένα από τα δείγματα, $\boldsymbol{x}_n$, $n = 1, 2, \ldots, N$. Κάθε φορά που ένα δείγμα κατηγοριοποιείται λανθασμένα, υλοποιείται μια διορθωτική ενημέρωση των παραμέτρων. Αλλιώς δεν απαιτείται κάποια ενέργεια. Όταν όλα τα δείγματα έχουν ελεγχθεί, λέμε ότι έχει ολοκληρωθεί μία εποχή (epoch). Αν δεν έχει επιτευχθεί σύγκλιση, όλα τα δείγματα παρουσιάζονται ξανά σε μια δεύτερη εποχή κ.ο.κ. Το παραπάνω είναι γνωστό ως pattern-by-pattern ή online μέθοδος.

Μετά από πεπερασμένο αριθμό εποχών, έχοντας επιλέξει κατάλληλα την τιμή της παραμέτρου μ_i, ο αλγόριθμος συγκλίνει σε λύση. Μπορούμε να συνοψίσουμε τον αλγόριθμο παρακάτω.

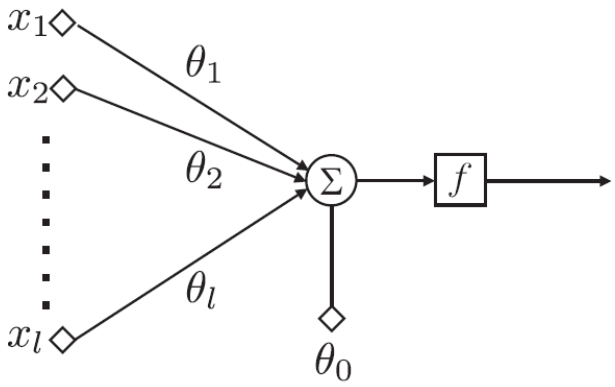
Ο online αλγόριθμος του Perceptron

- Initialization
  - $\boldsymbol{\theta}^{(0)} = 0$.
  - Select μ; usually it is set equal to one.
  - $i = 0$.
- **Repeat**; Each iteration corresponds to an epoch.

- $counter = 0$; Counts the number of updates per epoch.
- **For** $n = 1, 2, \ldots, N$, **Do**; For each epoch, all samples are presented.
  - **If** $(y_n \boldsymbol{x}_n^T \boldsymbol{\theta}^{(i-1)} \leq 0)$ **Then**
    - $i = i + 1$
    - $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} + \mu y_n \boldsymbol{x}_n$
    - $counter = counter + 1$
  - **End For**
- **Until** $counter = 0$

Όταν ο αλγόριθμος του perceptron έχει συγκλίνει, έχουμε τα βάρη (weights), $\theta_i,\ i = 1, 2, \ldots, l,$ των συνάψεων του διασυνδεδεμένου νευρώνα καθώς και τον bias όρο θ_0. Αυτά μπορούν, πλέον, να χρησιμοποιηθούν για την κατηγοριοποίηση άγνωστων προτύπων. Στην εικόνα 2.1 φαίνεται η αρχιτεκτονική του βασικού νευρώνα. Οι συνιστώσες (features), $x_i,\ i = 1, 2, \ldots, l,$ εφαρμόζονται στους κόμβους εισόδου (input nodes). Στη συνέχεια, κάθε feature πολλαπλασιάζεται με το αντίστοιχο (συναπτικό) βάρος και έπειτα προστίθεται ο bias όρος. Η έξοδος αυτή οδηγείται σε μια μη γραμμική συνάρτηση, f, γνωστή ως συνάρτηση ενεργοποίησης (activation function). Ανάλογα τη συνάρτηση αυτή, υπάρχουν διαφορετικά είδη νευρώνων. Στον κλασικό νευρώνα των McCulloch και Pitts η activation function είναι η Heaviside,

$$f(z) = \begin{cases} 1, & \alpha\nu\ z > 0 \\ 0, & \alpha\nu\ z \leq 0 \end{cases}$$



Εικόνα 2.1: Αρχιτεκτονική βασικού νευρώνα

2.3 Πολυστρωματικά Νευρωνικά Δίκτυα

Ένας νευρώνας αντιστοιχεί σε ένα υπερεπίπεδο

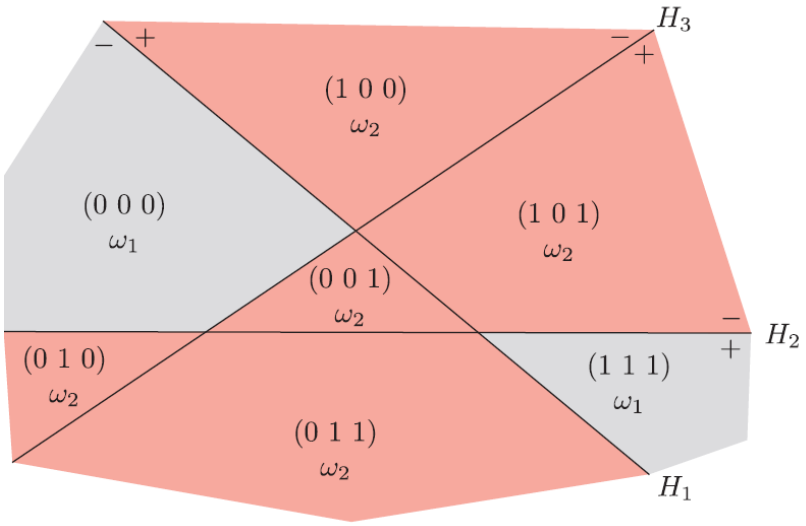
$$\boldsymbol{H}:\ \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_l x_l + \theta_0 = 0,$$

στο διανυσματικό χώρο της εισόδου. Ακόμα, η ταξινόμηση γίνεται μέσω μιας μη-γραμμικότητας (nonlinearity), η οποία δίνει 0 ή 1 ανάλογα σε ποια μεριά του $\boldsymbol{H}$ βρίσκεται το σημείο (εισόδου). Πώς, όμως, οι νευρώνες ενός δικτύου συνδέονται μεταξύ τους για να συνθέσουν μη γραμμικούς ταξινομητές (classifiers);

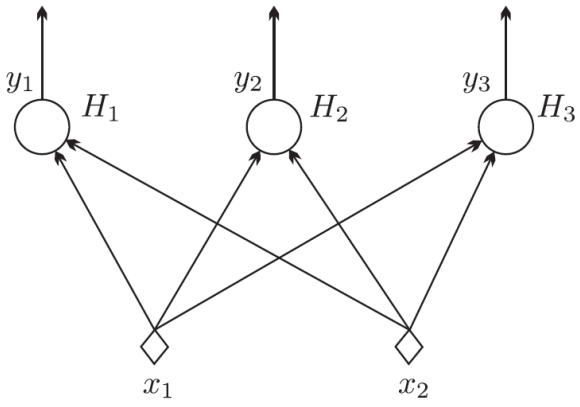
Μελετάμε, [3, σ 882], την περίπτωση που οι κλάσεις στον χώρο των προτύπων σχηματίζονται από ενώσεις πολυεδρικών περιοχών. Αυτό φαίνεται στην εικόνα 2.2, στην περίπτωση δισδιάστατου χώρου εισόδου. Οι πολυεδρικές περιοχές σχηματίζονται από τις τομές των ημι-χώρων που δημιουργούν τα επιμέρους υπερεπίπεδα (ευθείες για τον R^2). Υπάρχουν τρία υπερεπίπεδα, $\boldsymbol{H_1}$, $\boldsymbol{H_2}$, $\boldsymbol{H_3}$ τα οποία δημιουργούν εφτά πολυεδρικές περιοχές. Για κάθε ευθεία, ορίζουμε με $(+)$ και $(-)$ το θετικό και τον αρνητικό ημι-χώρο αντίστοιχα. Κάθε περιοχή περιγράφεται από μια τριάδα δυαδικών αριθμών ανάλογα με τη θέση της σε σχέση με τις ευθείες. Για παράδειγμα, η περιοχή (101), βρίσκεται στη $(+)$ μεριά του $\boldsymbol{H_1}$, στη $(-)$ μεριά του $\boldsymbol{H_2}$ και στη $(+)$ μεριά του $\boldsymbol{H_3}$.

Στόχος μας είναι ένα νευρωνικό δίκτυο ικανό να διαχωρίζει τα πρότυπα που ανήκουν στην κλάση ω_1 από αυτά που ανήκουν στην ω_2. Είναι προφανές ότι δεν υπάρχει υπερεπίπεδο (ευθεία) που να μπορεί να επιτύχει τον ζητούμενο διαχωρισμό του χώρου. Με άλλα λόγια, οι κλάσεις ω_1 και ω_2 δεν είναι γραμμικά διαχωρίσιμες και συνεπώς το απλό perceptron δεν είναι αρκετό για το πρόβλημα αυτό.

Στην εικόνα 2.3 φαίνονται τρεις νευρώνες που αντιστοιχούν στα τρία υπερεπίπεδα. Οι έξοδοι y_1, y_2, y_3 μαζί αναπαριστούν την περιοχή του χώρου στην οποία ανήκει το πρότυπο εισόδου. Έτσι, με τα συναπτικά βάρη σωστά ορισμένα, για ένα πρότυπο που ανήκει στην περιοχή (010) ο πρώτος νευρώνας θα δώσει 0, ο δεύτερος 1 και ο τρίτος 0.



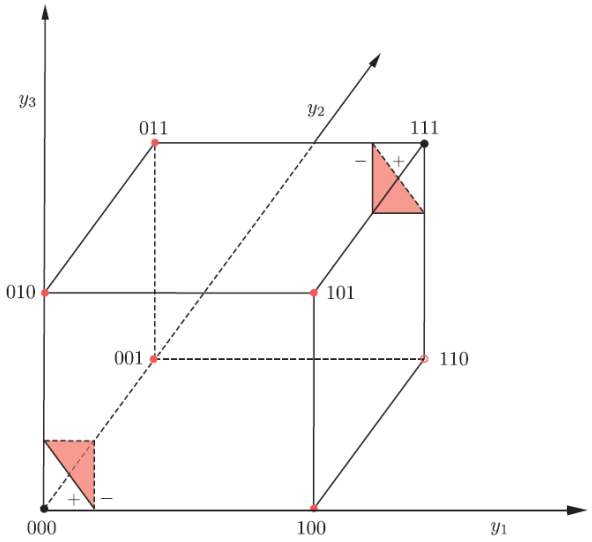
Εικόνα 2.2: Κλάσεις που σχηματίζονται από την ένωση πολυεδρικών περιοχών



Εικόνα 2.3: 1ο hidden layer κωδικοποιεί την περιοχή του χώρου στην οποία ανήκει το πρότυπο

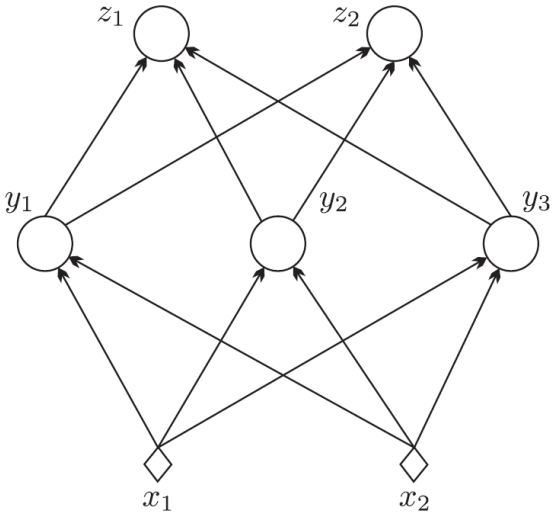
Ουσιαστικά, έχουμε κάνει μια αντιστοίχιση (mapping) του χώρου εισόδου (input feature space) σε έναν τρισδιάστατο χώρο. Πιο συγκεκριμένα, ο χώρος αυτός είναι οι κορυφές ενός μοναδιαίου κύβου στον R^3, όπως φαίνεται στην εικόνα 2.4. Παραμένουμε στον στόχο μας, που είναι να διαχωρίσουμε τον χώρο που ανήκουν οι δύο κλάσεις ώστε με την εκμάθηση των κατάλληλων βαρών, να επιτυγχάνεται σωστή ταξινόμηση των προτύπων.

Παρατηρούμε ότι στον νέο χώρο κάθε κορυφή είναι γραμμικά διαχωρίσιμη από όλες τις υπόλοιπες. Αυτό σημαίνει ότι μπορούμε να χρησιμοποιήσουμε έναν νευρώνα ο οποίος θα τοποθετήσει τη ζητούμενη κορυφή στη $(+)$ μεριά και όλους τις υπόλοιπες κορυφές στη $(-)$. Όπως φαίνεται στην εικόνα, χρειαζόμαστε να διαχωρίσουμε 2 κορυφές με 2 υπερεπίπεδα κάτι που σημαίνει πως θα έχουμε δύο νευρώνες στο 2ο hidden layer.



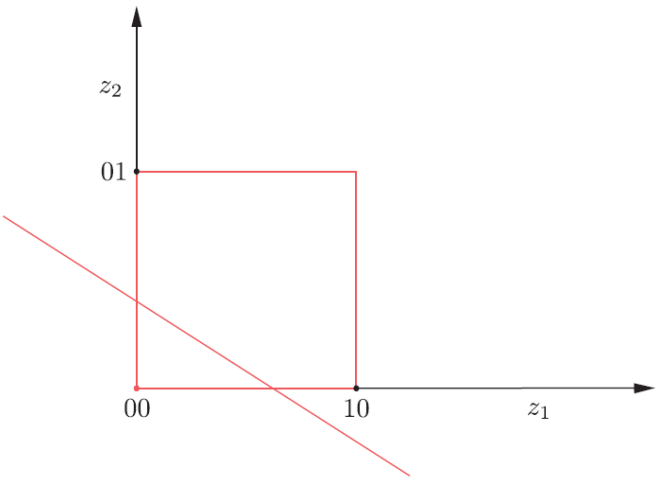
Εικόνα 2.4: Τρισδιάστατος χώρος στον οποίο αντιστοιχείται ο χώρος εισόδου μετά το 1ο layer

Στην εικόνα 2.5 παρουσιάζεται το 2ο στρώμα που αποτελείται από τους νευρώνες z_1 και z_2. Σημειώνεται ότι ο z_1 θα δώσει 1 μόνο αν το πρότυπο προέρχεται από την περιοχή (000) και αντίστοιχα ο z_2 μόνο αν προέρχεται από την (111). Σε όποια άλλη περίπτωση οι δύο νευρώνες δίνουν 0 στην έξοδό τους.



Εικόνα 2.5: Το Νευρωνικό δίκτυο με την προσθήκη του 2ου hidden layer

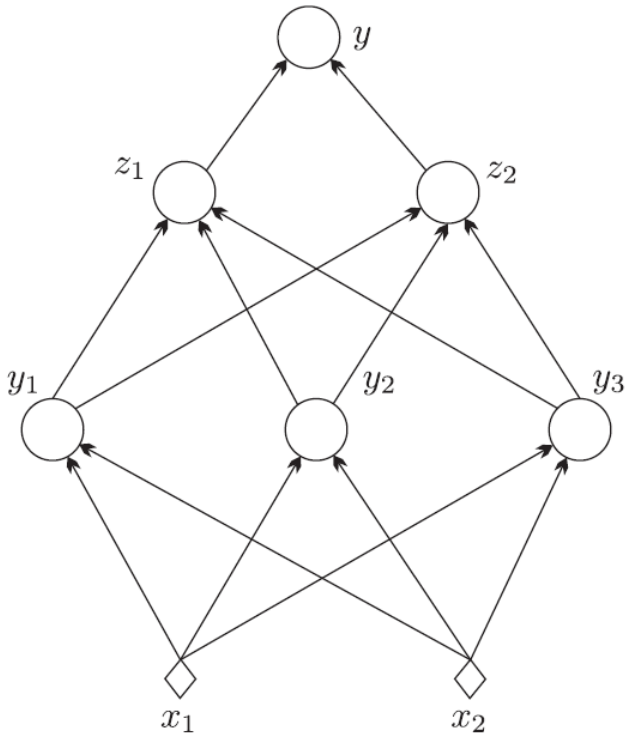
Το 2ο hidden layer υλοποιεί μια νέα αντιστοίχιση αυτή τη φορά σε έναν διδιάστατο χώρο. Συγκεκριμένα, ο νέος αυτός χώρος είναι οι κορυφές ενός μοναδιαίου τετραγώνου, εικόνα 2.6.



Εικόνα 2.6: Διδιάστατος χώρος μετά το 2ο hidden layer

Παρατηρούμε ότι όλα τα πρότυπα που ανήκουν στην κλάση ω_2 αντιστοιχούν στην κορυφή (00) ενώ εκείνα που ανήκουν στην κλάση ω_1 σε μία από τις (01) και (10). Αυτό είναι εξαιρετικά ενδιαφέρον. Με κατάλληλες αντιστοιχίσεις επιτύχαμε τον μετασχηματισμό του αρχικού μη-γραμμικά διαχωρίσιμου προβλήματος σε ένα γραμμικά διαχωρίσιμο.

Το τελευταίο που απαιτείται είναι ένας νευρώνας εξόδου (output neuron) για να υλοποιήσει την τελική απόφαση για την ταξινόμηση της εισόδου. Στην εικόνα 2.7 φαίνεται το τελικό νευρωνικό δίκτυο.



Εικόνα 2.7: Τελικό νευρωνικό δίκτυο μαζί με το νευρώνα εξόδου

Το παραπάνω δίκτυο ονομάζεται feed-forward αφού η ροή της πληροφορίας ρέει από την είσοδο προς την έξοδό του. Αποτελείται από το input layer (συνήθως δεν προσμετράται στα layers καθώς δεν γίνεται κάποια επεξεργασία των δεδομένων σε αυτό), τα 2 hidden layers και από το output layer. Λέμε ότι έχουμε ένα feed-forward network με 3 layers.

Η παραπάνω μελέτη έγινε σε ένα συγκεκριμένο πρόβλημα δύο μη γραμμικά διαχωρίσιμων κλάσεων που ανήκαν σε ενώσεις πολυεδρικών περιοχών. Η γενίκευση του προβλήματος αυτού για περισσότερες από δύο κλάσεις είναι άμεση, καθώς η μόνη προσθήκη που πρέπει να γίνει είναι στο output layer με την ύπαρξη περισσότερων νευρώνων για την ταξινόμηση σε περισσότερες κλάσεις. Για παράδειγμα, αν είχαμε 5 κλάσεις, 2 νευρώνες δεν θα ήταν αρκετοί γιατί θα μας έδιναν μέχρι 4 διαφορετικούς συνδυασμούς ως έξοδο του νευρωνικού δικτύου. Συνεπώς, το δίκτυο μας θα είχε τότε 3 νευρώνες στο output layer κ.ο.κ.

Στη ζωή, βέβαια, τα προβλήματα πολύ σπάνια περιγράφονται από κλάσεις που ανήκουν σε πολυεδρικές περιοχές και πολλές φορές υπάρχει επικάλυψη μεταξύ κλάσεων. Εκεί, χρειαζόμαστε μια εκπαιδευτική διαδικασία στηριγμένη - κατά βάση - στη συνάρτηση κόστους.

Αυτό που έχει μεγάλη σημασία να κρατήσουμε από την παραπάνω μελέτη είναι η δομή του multilayer νευρωνικού δικτύου και οι μετασχηματισμοί στην αναπαράσταση των δεδομένων που συντελούνται από την είσοδο προς την έξοδό του. Κάθε layer είναι ένα νέο mapping, ιδανικά με περισσότερη πληροφορία (ως προς το πρόβλημα μας) από το προηγούμενο, ώσπου στο τελευταίο layer το πρόβλημα μας να έχει απλοποιηθεί τόσο που να οδηγηθούμε σε λύση.

2.4 Αλγόριθμος Backpropagation με Gradient Descent

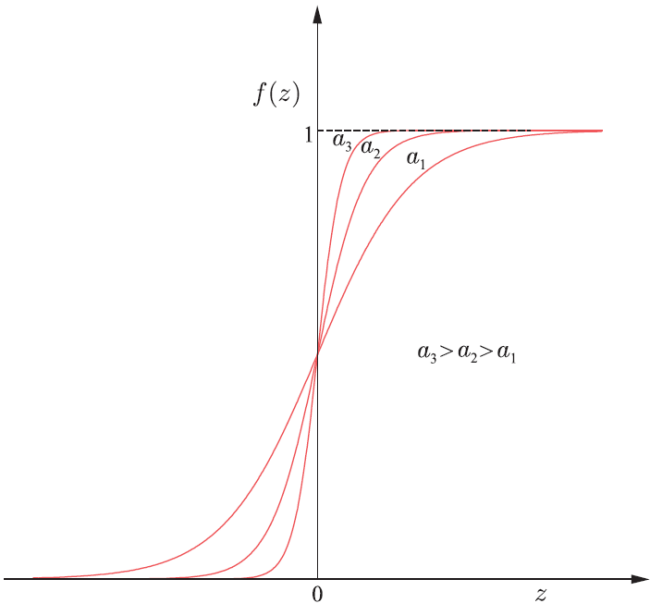
Ένα feedforward νευρωνικό δίκτυο αποτελείται από πλήθος layers νευρώνων και κάθε νευρώνας έχει ένα σύνολο από συναπτικά βάρη και τον bias όρο του. Από αυτή τη σκοπιά, ένα νευρωνικό δίκτυο υλοποιεί μια μη γραμμική συνάρτηση, $\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x})$, όπου $\boldsymbol{\theta}$ είναι όλα τα βάρη και biases που υπάρχουν στο δίκτυο [3, σ 886]. Οπότε, η εκπαίδευση ενός νευρωνικού δικτύου δείχνει να μην έχει κάποια διαφοροποίηση από την εκπαίδευση οποιουδήποτε παραμετρικού μοντέλου πρόβλεψης. Αυτά που χρειάζονται είναι (α) ένα σύνολο δειγμάτων εκπαίδευσης, (β) μια συνάρτηση απωλειών (loss function) και (γ) ένα επαναληπτικό σχήμα, για παράδειγμα gradient descent, για την ελαχιστοποίηση των συνολικών απωλειών,

$$J(\boldsymbol{\theta}) = \sum_{n=1}^{N} L(y_n, f_{\boldsymbol{\theta}}(\boldsymbol{x}_n))$$

Η δυσκολία στην εκπαίδευση ενός νευρωνικού οφείλεται στην πολυστρωματική δομή του που περιπλέκει τον υπολογισμό των μερικών παραγώγων (gradients), απαραίτητων για τη βελτιστοποίηση. Επιπλέον, ο McCulloch-Pitts νευρώνας περιέχει τη συνάρτηση Heaviside για ενεργοποίηση η οποία δεν είναι συνεχής και συνεπώς ούτε παραγωγίσιμη. Ένα πρώτο βήμα για την ανάπτυξη ενός πρακτικού αλγορίθμου εκπαίδευσης του νευρωνικού δικτύου είναι η αντικατάσταση της Heaviside με μια παραγωγίσιμη συνάρτηση που αποτελεί προσέγγισή της.

Μια πολύ συχνά χρησιμοποιούμενη συνάρτηση ενεργοποίησης είναι η λογιστική σιγμοειδής συνάρτηση (logistic sigmoid function), εικόνα 2.8, η οποία ορίζεται ως

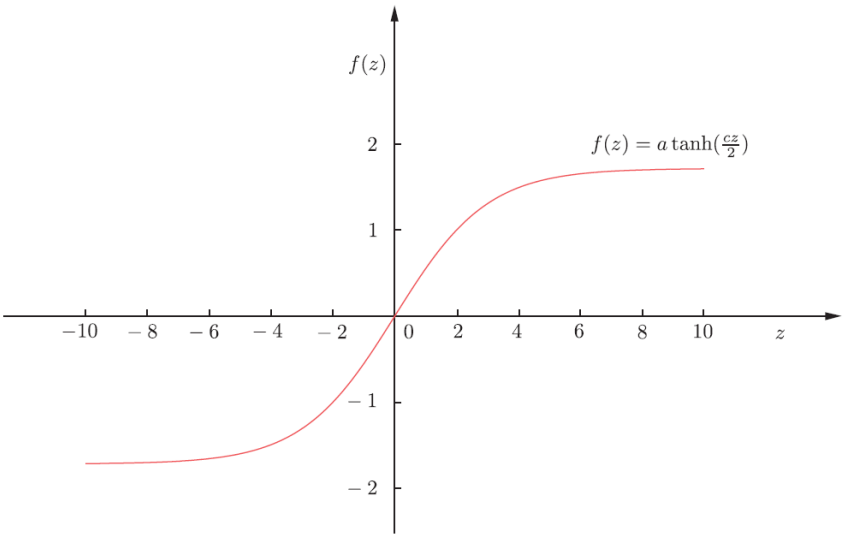
$$f(z) = \sigma(z) \coloneqq \frac{1}{1 + \exp(-az)}$$



Εικόνα 2.8: Λογιστική σιγμοειδής συνάρτηση για διάφορες τιμές της παραμέτρου α

Μια άλλη συνάρτηση που χρησιμοποιείται για τον ίδιο σκοπό είναι η συνάρτηση υπερβολικής εφαπτομένης με παραμέτρους, εικόνα 2.9,

$$f(z) = \alpha \tanh\left(\frac{cz}{2}\right)$$



Εικόνα 2.9: Η συνάρτηση υπερβολικής εφαπτομένης για συγκεκριμένες τιμές α, c

Η μέθοδος Gradient Descent

Έχοντας χρησιμοποιήσει μια παραγωγίσιμη συνάρτηση ενεργοποίησης, είμαστε έτοιμοι να αναπτύξουμε την επαναληπτική μέθοδο gradient descent (καθόδου κλίσης) για την ελαχιστοποίηση της συνάρτησης κόστους. [3, σ 887]

Έστω $(\boldsymbol{y}_n, \boldsymbol{x}_n),\ n = 1, 2, \dots, N,$ το σύνολο των δειγμάτων εκπαίδευσης. Σημειώνεται ότι υποθέσαμε πολλαπλές μεταβλητές εξόδου, για αυτό το $\boldsymbol{y}$ συμβολίστηκε ως διάνυσμα. Υποθέτουμε ότι το δίκτυο συντίθεται από L στρώματα, $L-1$ hidden layers και το output layer. Κάθε layer αποτελείται από $k_r,\ r = 1, 2, \dots, L,$ νευρώνες. Έτσι, τα διανύσματα εξόδου (output vectors) θα είναι,

$$\boldsymbol{y}_n = \left[y_{n1}, y_{n2}, \dots, y_{nk_L}\right]^T \in R^{k_L}, \qquad n = 1, 2, \dots, N.$$

Για να χρησιμοποιηθούν στις παραγώγους, συμβολίζουμε το πλήθος των κόμβων εισόδου (input nodes) με k_0, για το οποίο ισχύει ότι $k_0 = l,$ όπου l η διάσταση του χώρου εισόδου (input feature space).

Έστω $\boldsymbol{\theta}_j^r$ τα συναπτικά βάρη του $j-οστού$ νευρώνα που βρίσκεται στο $r-οστό$ layer, με $j = 1, 2, \dots, k_r$ και $r = 1, 2, \dots, L,$ ενώ ο bias όρος συμπεριλαμβάνεται στο $\boldsymbol{\theta}_j^r$,

$$\boldsymbol{\theta}_j^r \coloneqq [\theta_{j0}^r, \theta_{j1}^r, \dots, \theta_{jk_{r-1}}^r]^{\mathrm{T}}$$

Τα συναπτικά βάρη συνδέουν τον νευρώνα με όλους τους νευρώνες του k_{r-1} layer. Το βασικό επαναληπτικό βήμα για τη μέθοδο gradient descent είναι,

$$\boldsymbol{\theta}_j^r(new) = \boldsymbol{\theta}_j^r(old) + \Delta\boldsymbol{\theta}_j^r,$$

όπου,

$$\Delta\boldsymbol{\theta}_j^r = -\mu \left.\frac{\partial J}{\partial \boldsymbol{\theta}_j^r}\right|_{\boldsymbol{\theta}_j^r(old)}$$

Η παράμετρος μ ορίζεται από τον χρήστη και σχετίζεται με το μήκος-βήματος ενώ J είναι η συνάρτηση κόστους.

Υπολογισμός των gradients

Έστω z_{nj}^r συμβολίζει την έξοδο του γραμμικού αθροιστή (ακριβώς πριν τη συνάρτηση ενεργοποίησης) του $j-οστού$ νευρώνα στο $r-οστό$ layer τη χρονική στιγμή n, δηλαδή όταν το πρότυπο $\boldsymbol{x}_n$ εφαρμόζεται στους input nodes. Τότε μπορούμε να γράψουμε ότι,

$$z_{nj}^r = \sum_{m=1}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} + \theta_{j0}^r = \sum_{m=0}^{k_{r-1}} \theta_{jm}^r y_{nm}^{r-1} = \boldsymbol{\theta}_j^{rT} \boldsymbol{y}_n^{r-1}$$

όπου από ορισμό,

$$\boldsymbol{y}_n^{r-1} \coloneqq \left[1, y_{n1}^{r-1}, \dots, y_{nk_{r-1}}^{r-1}\right]^T$$

και $y_{n0}^r \equiv 1, \forall\, r, n$. Για τους νευρώνες του output layer, $r = L,\ y_{nm}^L = \hat{y}_{nm}, m = 1, 2, \dots, k_L,$ ενώ για $r = 1$, input layer, $y_{nm}^0 = x_{nm}, m = 1, 2, \dots, k_0$.

Οπότε, μπορούμε τώρα να γράψουμε

$$\frac{\partial J_n}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\frac{\partial z_{nj}^r}{\partial \boldsymbol{\theta}_j^r} = \frac{\partial J_n}{\partial z_{nj}^r}\boldsymbol{y}_n^{r-1}$$

Με τον ορισμό

$$\delta_{nj}^r := \frac{\partial J_n}{\partial z_{nj}^r}$$

Παίρνουμε

$$\Delta\boldsymbol{\theta}_j^r = -\mu\sum_{n=1}^{N}\delta_{nj}^r\,\boldsymbol{y}_n^{r-1}, \qquad r = 1, 2, \dots, L.$$

Υπολογισμός του $\boldsymbol{\delta_{nj}^r}$

Εδώ είναι το σημαντικότερο σημείο για τον αλγόριθμο backpropagation. Για τον υπολογισμό των gradients, δ_{nj}^r, ξεκινάμε από το τελευταίο layer, $r = L$, και συνεχίζουμε προς τα πίσω μέχρι το $r = 1$.

1. Για $r = L$

$$\delta_{nj}^L = \frac{\partial J_n}{\partial z_{nj}^L}$$

υπολογισμός που γίνεται ευθέως. Για παράδειγμα, αν έχουμε για συνάρτηση απωλειών τη συνάρτηση τετραγώνου σφάλματος,

$$J_n = \frac{1}{2}\sum_{k=1}^{k_L}(f(z_{nk}^L) - \boldsymbol{y}_{nk})^2$$

τότε,

$$\delta_{nj}^L = \left(\hat{\boldsymbol{y}}_{nj} - \boldsymbol{y}_{nj}\right)f'\left(z_{nj}^L\right), \qquad j = 1, 2, \dots, k_L,$$

τα οποία είναι όλα γνωστά για τον υπολογισμό του gradient.

2. Για $r < L$, εφαρμόζουμε τον κανόνα αλυσίδας για την παραγώγιση,

$$\delta_{nj}^{r-1} = \frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r}\frac{\partial J_n}{\partial z_{nk}^r}\frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}}$$

ή

$$\frac{\partial J_n}{\partial z_{nj}^{r-1}} = \sum_{k=1}^{k_r}\delta_{nk}^r\frac{\partial z_{nk}^r}{\partial z_{nj}^{r-1}}$$

Όμως,

$$\frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} = \frac{\partial\left(\sum_{m=0}^{k_{r-1}} \theta_{km}^{r} y_{nm}^{r-1}\right)}{\partial z_{nj}^{r-1}}$$

ή

$$\frac{\partial z_{nk}^{r}}{\partial z_{nj}^{r-1}} = \theta_{kj}^{r} f'\left(z_{nj}^{r-1}\right)$$

Συνδυάζοντας τις παραπάνω φτάνουμε στον επαναληπτικό κανόνα

$$\delta_{nj}^{r-1} = \left(\sum_{k=1}^{k_r} \delta_{nk}^{r} \theta_{kj}^{r}\right) f'\left(z_{nj}^{r-1}\right), \qquad j = 1, 2, \ldots, k_{r-1}.$$

Από τον κανόνα αυτόν είναι εμφανές πως το σφάλμα, κατ' επέκταση τα gradients, μεταδίδονται από το τελευταίο layer σε όλα προηγούμενα, κάτι που δικαιολογεί το όνομα του αλγορίθμου backpropagation (διάδοσης του σφάλματος προς τα πίσω).

Ο αλγόριθμος backpropagation με gradient descent

• Initialization

  • Initialize all weights and biases randomly with small, but not very small, values.

  • Select step size μ.

  • Set $y_{nj}^{0} = x_{nj}, j = 1, 2, \ldots, k_0 = l, n = 1, 2, \ldots, N$.

• **Repeat**; Each repetition completes an epoch.

  • **For** $n = 1, 2, \ldots, N$, **Do**

    - **For** $r = 1, 2, \ldots, L$, **Do**; Forward computations.

      • **For** $j = 1, 2, \ldots, k_r$, **Do**

        Compute z_{nj}^{r}.

        Compute $y_{nj}^{r} = f(z_{nj}^{r})$.

      • **End For**

    - End For

    - For $j = 1, 2, \ldots, k_L$, **Do**

      • Compute δ_{nj}^{L} .

    - End For

    - For $r = L, L-1, \ldots, 2$, **Do**; Backward computations.

      • **For** $j = 1, 2, \ldots, k_r$, **Do**

Compute δ_{nj}^{r-1}.

• **End For**

- End For

• **End For**

• **For** $r = 1, 2, \ldots, L$, **Do**; Update the weights.

- For $j = 1, 2, \ldots, k_r$, **Do**

• Compute $\Delta\boldsymbol{\theta}_j^r$.

• $\boldsymbol{\theta}_j^r = \boldsymbol{\theta}_j^r + \Delta\boldsymbol{\theta}_j^r$

- End For

• **End For**

• **Until** a stopping criterion is met.

Σημειώνουμε ότι κομβικής σημασίας είναι η επιλογή του step-size μ. Θα πρέπει να είναι μικρό για να εγγυάται τη σύγκλιση του αλγορίθμου, αλλά όχι πολύ μικρό για να μην έχει αντίκρισμα στην ταχύτητα σύγκλισης. Επιπλέον, παραπάνω οι τιμές των βαρών ενημερώνονται στο τέλος κάθε εποχής. Ουσιαστικά, περιγράφεται η batch λειτουργία του αλγορίθμου. Μπορούμε, φυσικά, να εφαρμόσουμε pattern-by-pattern λειτουργία με ενημέρωση των βαρών σε κάθε βήμα. Η διαφορά τους έχει να κάνει με το άθροισμα των gradients για όλα τα πρότυπα στην πρώτη περίπτωση, κάτι που δεν συμβαίνει στη δεύτερη. Το άθροισμα αυτό πιθανότατα θα αλληλοαναιρέσει κάποια σφάλματα. Από την άλλη, στην περίπτωση της pattern-by-pattern λειτουργίας υπάρχει μεγάλη τυχαιότητα στις διορθωτικές κινήσεις που θα κάνει ο αλγόριθμος. Στην πράξη, συνήθως, χρησιμοποιείται η mini-batch λειτουργία κατά την οποία η ενημέρωση των βαρών γίνεται κάθε $N_1 < N$ δείγματα.

2.5 Επιλογή συνάρτησης κόστους και μονάδων εξόδου

Ένα πολύ σημαντικό ζήτημα στο σχεδιασμό του νευρωνικού δικτύου, [7, σσ 178–182], είναι η επιλογή της συνάρτησης κόστους. Οι συναρτήσεις αυτές για τα νευρωνικά δίκτυα είναι λίγο έως πολύ ίδιες με εκείνες για οποιοδήποτε παραμετρικό μοντέλο.

Στις περισσότερες περιπτώσεις, το παραμετρικό μοντέλο ορίζει μια κατανομή $p(\boldsymbol{y} \mid \boldsymbol{x}; \boldsymbol{\theta})$ και συνεπώς χρησιμοποιούμε την αρχή της μέγιστης πιθανοφάνειας (maximum likelihood). Αυτό σημαίνει ότι η συνάρτηση κόστους θα βασίζεται στην εντροπία (cross-entropy) ανάμεσα στα δεδομένα εκπαίδευσης και στις προβλέψεις του μοντέλου.

Υιοθετώντας, [3, σ 896], ως στόχους τις τιμές $0, 1$, η πραγματική τιμή και η πρόβλεψη, $\boldsymbol{y}_{nm}, \hat{\boldsymbol{y}}_{nm}$, $n = 1, 2, \ldots, N$, $m = 1, 2, \ldots, k_L$, μπορούν να θεωρηθούν πιθανότητες και η συνάρτηση κόστους cross-entropy είναι,

$$J = -\sum_{n=1}^{N}\sum_{k=1}^{k_L}\left(y_{nk}\ln\hat{y}_{nk} + (1 - y_{nk})\ln(1 - \hat{y}_{nk})\right)$$

η οποία παίρνει την ελάχιστη τιμή της όταν $\boldsymbol{y}_{nm} = \hat{\boldsymbol{y}}_{nm}$, που για δυαδικούς στόχους είναι μηδέν.

Σε ένα classification πρόβλημα, κατά την εκπαίδευση, το $\boldsymbol{y}_{nm}$ θα έχει 0 σε όλες τις κλάσεις εκτός από εκείνη στην οποία ανήκει όπου θα έχει 1. Για να έχουμε και στην πρόβλεψη $\widehat{\boldsymbol{y}}_{nm}$ κατανομή πιθανοτήτων που να αθροίζει στη μονάδα χρησιμοποιούμε στο output layer την softmax συνάρτηση ενεργοποίησης,

$$\widehat{\boldsymbol{y}}_{nk} = \frac{\exp(z_{nk}^{L})}{\sum_{m=1}^{k_L} \exp(z_{nm}^{L})}$$

2.6 Συναρτήσεις Ενεργοποίησης των hidden units

Στην παράγραφο 2.4 παρουσιάστηκαν συνοπτικά δύο κλασικές συναρτήσεις ενεργοποίησης, η λογιστική σιγμοειδής και η συνάρτηση υπερβολικής εφαπτομένης. Λόγω της σημασίας που έχει η συνάρτηση ενεργοποίησης (activation function) για τα νευρωνικά δίκτυα, καλό είναι να μελετηθεί με περισσότερη λεπτομέρεια. Δεν είναι υπερβολή να πούμε ότι στις συναρτήσεις ενεργοποίησης χτυπάει η καρδιά του νευρωνικού δικτύου καθώς εκεί είναι το σημείο όπου εισάγεται η μη-γραμμικότητα στη συμπεριφορά του, στοιχείο απαραίτητο για την αντιμετώπιση των προβλημάτων τα οποία καλείται να επιλύσει.

Η σωστή επιλογή των κρυφών μονάδων (hidden units) είναι ένα άκρως ενεργό ερευνητικό πεδίο, το οποίο δεν έχει ακόμα αρκετές, σαφώς καθορισμένες, θεωρητικές αρχές. [7, σ 191]

Τα Rectified linear units, αλλιώς Rectifier (Ανορθωτής), γνωστά και με το ακρωνύμιο ReLU, είναι μια εξαιρετική αρχική επιλογή για τα hidden units. Άλλοι τύποι hidden units είναι επίσης διαθέσιμοι. Η επιλογή των κατάλληλων μονάδων γίνεται με τη διαδικασία δοκιμής και σφάλματος (trial and error) μέσω της αξιολόγησης της απόδοσης του μοντέλου στο validation set[1].

2.6.1 Rectified Linear Units

Οι γραμμικές μονάδες ανόρθωσης χρησιμοποιούν τη συνάρτηση ενεργοποίησης,

$$f(z) = \max\{0, z\}$$

Οι ReLU είναι εύκολο να βελτιστοποιηθούν γιατί είναι παρόμοιες με τις γραμμικές μονάδες. Η μόνη διαφορά τους είναι ότι οι ReLU δίνουν μηδέν για οποιαδήποτε μη θετική τιμή. Αυτό κάνει τις μερικές παραγώγους μιας ReLU να παραμένουν μεγάλες όταν η μονάδα είναι ενεργή. Οι μερικές παράγωγοι δεν είναι απλά μεγάλες, είναι και συνεχείς. Η δεύτερη παράγωγος της συνάρτησης είναι 0 σχεδόν παντού και η πρώτη παράγωγος είναι 1 όπου η μονάδα είναι ενεργή. Αυτό σημαίνει ότι η κατεύθυνση της κλίσης (gradient) είναι πολύ πιο χρήσιμη για τη μάθηση απ' ό,τι θα ήταν με οποιαδήποτε συνάρτηση ενεργοποίησης εισήγαγε δευτέρας-τάξης επιδράσεις. [7, σ 193]

Οι ReLU περιγράφονται συνολικά από την ακόλουθη σχέση,

$$\boldsymbol{h} = f(\boldsymbol{W}^T\boldsymbol{x} + \boldsymbol{b})$$

[1] Σύνολο από δείγματα που δεν χρησιμοποιούνται στην εκπαίδευση, αλλά στη ρύθμιση (tuning) υπερ-παραμέτρων (hyper-parameters) του αλγορίθμου εν γένει, μέσω της αξιολόγησής του (evaluation).

Κατά την αρχικοποίηση των παραμέτρων, είναι καλή πρακτική ο ορισμός των στοιχείων του $\boldsymbol{b}$ σε μια μικρή θετική τιμή, όπως 0.1. Αυτό κάνει πολύ πιθανό οι ReLU να είναι αρχικά ενεργές, για τις περισσότερες εισόδους, και οι μερικές παράγωγοι να κυκλοφορήσουν στο δίκτυο.

Ένα μειονέκτημα των rectified linear units είναι ότι δεν μπορούν να μάθουν, με μεθόδους που βασίζονται σε gradient, όταν η ενεργοποίησή τους είναι μηδενική. Ένα πλήθος γενικεύσεων των ReLU έχουν αναπτυχθεί για να εξασφαλίζουν τη δυνατότητα μάθησης σε οποιαδήποτε περίπτωση. Κάποιες από αυτές εισάγουν μια μη μηδενική κλίση στη συνάρτηση ενεργοποίησης για αρνητικές τιμές εισόδου (Absolute value rectification, leaky ReLU, parametric ReLU), ενώ άλλες γενικεύουν ακόμα περισσότερο τις ReLU παίρνοντας, για παράδειγμα, ομάδες εισόδων αντί για μεμονωμένα παραδείγματα (Maxout units). Περισσότερες πληροφορίες μπορούν να βρεθούν εδώ [7, σ 194].

Οι ReLU και όλες οι γενικεύσεις τους βασίζονται στην αρχή ότι τα μοντέλα είναι ευκολότερο να βελτιστοποιηθούν αν η συμπεριφορά τους είναι πιο κοντά σε γραμμική. Η ίδια γενική αρχή, της χρησιμοποίησης γραμμικής συμπεριφοράς για την επίτευξη ευκολότερης βελτιστοποίησης, εφαρμόζεται και σε άλλα πεδία της βαθιάς μηχανικής μάθησης. Τα Recurrent Networks μπορούν να μάθουν από ακολουθίες και να παράξουν ακολουθίες καταστάσεων και εξόδων. Κατά την εκπαίδευση, γίνεται μετάδοση της πληροφορίας σε διάφορα χρονικά βήματα, κάτι που διευκολύνεται όταν υπεισέρχονται αρκετοί γραμμικοί υπολογισμοί (με σχετικές παραγώγους που παίρνουν την τιμή 1). Τα Recurrent Neural Networks, καθώς και οι καλύτεροι εκπρόσωποί τους, τα LSTMs, αναλύονται στο κεφάλαιο 6 της εργασίας.

2.6.2 Λογιστική Σιγμοειδής και Υπερβολική Εφαπτομένη

Πριν την εμφάνιση των ReLU, τα περισσότερα νευρωνικά δίκτυα χρησιμοποιούσαν τη λογιστική σιγμοειδή ως συνάρτηση ενεργοποίησης,

$$f(z) = \sigma(z)$$

ή τη συνάρτηση υπερβολικής εφαπτομένης,

$$f(z) = tanh(z).$$

Οι συναρτήσεις αυτές συνδέονται αρκετά αφού $tanh(z) = 2\sigma(2z) - 1$.

Έχουμε δει τις δύο συναρτήσεις καθώς και τις γραφικές τους παραστάσεις στην παράγραφο 2.4. Σε αντίθεση με τις γραμμικές μονάδες (ή τις μερικώς γραμμικές όπως οι ReLU), οι σιγμοειδείς παρουσιάζουν κορεσμό σχεδόν σε όλο το πεδίο ορισμού τους. Κορένονται σε μια υψηλή τιμή όταν το z είναι αρκετά θετικό, σε μια χαμηλή τιμή όταν είναι αρνητικό, και παρουσιάζουν υψηλή ευαισθησία μόνο όταν το z είναι γύρω από το 0. Ο διευρυμένος κορεσμός των σιγμοειδών μονάδων μπορεί να καταστήσει τη μάθηση, μέσω των gradients, αρκετά δυσκολότερη. Για τον λόγο αυτό, έχουν πάψει να χρησιμοποιούνται στα hidden layers ενός νευρωνικού δικτύου. Μπορούν, όμως, να χρησιμοποιηθούν στο output layer σε συνδυασμό με μια συνάρτηση κόστους που θα αναιρεί τον κορεσμό των σιγμοειδών μονάδων. [7, σ 195]

Όταν μια σιγμοειδούς-τύπου συνάρτηση ενεργοποίησης πρέπει να χρησιμοποιηθεί, η συνάρτηση υπερβολικής εφαπτομένης αποδίδει συνήθως καλύτερα από τη λογιστική σιγμοειδής. Προσομοιάζει την ταυτοτική συνάρτηση, $f(z) = z$, πολύ καλύτερα αφού $\tanh(0) = 0$ ενώ $\sigma(0) = 1/2$. Επειδή η υπερβολική εφαπτομένη είναι παρόμοια με την

ταυτοτική συνάρτηση κοντά στο 0, η εκπαίδευση ενός νευρωνικού δικτύου με 2 hidden layers,

$$\hat{y} = \boldsymbol{w}^{\boldsymbol{T}} \tanh(\boldsymbol{U}^{T} \tanh(\boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{x}))$$

μοιάζει με την εκπαίδευση του γραμμικού μοντέλου,

$$\hat{y} = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{U}^{T} \boldsymbol{V}^{\boldsymbol{T}} \boldsymbol{x}$$

όσο οι ενεργοποιήσεις του δικτύου κρατιόνται σε μικρές τιμές. Αυτό κάνει την εκπαίδευση του δικτύου με hidden units υπερβολικής εφαπτομένης ευκολότερη.

Άλλες συναρτήσεις ενεργοποίησης, που χρησιμοποιούνται λιγότερο συχνά, είναι η συνάρτηση ακτινικής βάσης (Radial Basis Function), η συνάρτηση Softplus και η σκληρή υπερβολική εφαπτομένη (Hard $\tanh$). Περισσότερες πληροφορίες μπορεί κάποιος να αναζητήσει εδώ [7, σ 197].

2.7 Ιδιότητα Καθολικής Προσέγγισης των Νευρωνικών Δικτύων

Ένα γραμμικό μοντέλο που αντιστοιχίζει τα features εισόδου σε ένα διάνυσμα εξόδου μέσω πολλαπλασιασμού πινάκων, μπορεί εξορισμού να αναπαραστήσει μόνο γραμμικές συναρτήσεις. Δυστυχώς, συχνά επιθυμούμε να μάθουμε μη-γραμμικές συναρτήσεις.

Σε πρώτη ανάγνωση θα υποθέταμε ότι είναι απαραίτητο να σχεδιάσουμε ένα ειδικό μη-γραμμικό μοντέλο για να μάθουμε μια μη-γραμμική σχέση μεταξύ εισόδου και εξόδου. Ευτυχώς, τα νευρωνικά δίκτυα με τουλάχιστον 1 κρυφό στρώμα και 1 στρώμα εξόδου παρέχουν τη δυνατότητα προσέγγισης οποιασδήποτε συνάρτησης. [7, σ 198]

Θεώρημα Καθολικής Προσέγγισης (Universal Approximation Theorem)

Έστω το διστρωματικό δίκτυο με ένα κρυφό στρώμα και έναν μοναδικό γραμμικό νευρώνα εξόδου. Η έξοδός του γράφεται

$$\hat{g}(\boldsymbol{x}) = \sum_{k=1}^{K} \theta_{\boldsymbol{k}}^{\boldsymbol{o}} f\left(\boldsymbol{\theta}_{\boldsymbol{k}}^{\boldsymbol{hT}} \boldsymbol{x}\right) + \theta_{\boldsymbol{0}}^{\boldsymbol{o}}$$

όπου $\boldsymbol{\theta}_{\boldsymbol{k}}^{\boldsymbol{h}}$ τα συναπτικά βάρη και ο όρος bias του k-οστού κρυφού νευρώνα ενώ ο δείκτης «o» αναφέρεται στον νευρώνα εξόδου.

Έστω, ακόμα, ότι $g(\boldsymbol{x})$ συνεχής συνάρτηση ορισμένη σε ένα συνεκτικό υποσύνολο του $S \subset \mathbb{R}^{l}$ και οποιοδήποτε $\epsilon > 0$. Τότε υπάρχει διστρωματικό δίκτυο με $K(\epsilon)$ κρυφούς νευρώνες όπως αυτό της παραπάνω εξίσωσης, τέτοιο ώστε

$$|g(\boldsymbol{x}) - \hat{g}(\boldsymbol{x})| < \epsilon, \forall \boldsymbol{x} \in S.$$

Το θεώρημα αναφέρει ότι ένα διστρωματικό νευρωνικό δίκτυο είναι αρκετό για να προσεγγίσει οποιαδήποτε συνεχή συνάρτηση. Μπορεί, δηλαδή, να υλοποιήσει οποιαδήποτε μη-γραμμική διάκριση περιοχών για ένα πρόβλημα ταξινόμησης (classification) ή να αναπτύξει οποιαδήποτε μη-γραμμική συνάρτηση για την πρόβλεψη κάποιας τιμής σε ένα κλασικό πρόβλημα παλινδρόμησης (regression). Αυτό, όμως, που δεν λέει το θεώρημα είναι πόσο μεγάλο θα πρέπει να είναι το δίκτυο αυτό όσον αφορά το πλήθος των νευρώνων του κρυφού στρώματος. Μπορεί να απαιτείται ένα υπερβολικά μεγάλο δίκτυο για να είναι εφικτή μια αρκετά καλή προσέγγιση. Εδώ είναι

που η χρήση περισσότερων στρωμάτων μπορεί να αποδειχτεί προσοδοφόρα. Χρησιμοποιώντας περισσότερα layers, το συνολικό πλήθος των νευρώνων που απαιτούνται μπορεί να είναι αρκετά μικρότερο. [3, σσ 899–900]

2.8 Deep Learning

Τα παραπάνω μας έχουν περιγράψει σε καλό βαθμό πως δομούνται και λειτουργούν τα νευρωνικά δίκτυα. Από το αρχικό μοντέλο του νευρώνα και το Perceptron, στα πολυστρωματικά νευρωνικά δίκτυα και στον αλγόριθμο του backpropagation με gradient descent για την ελαχιστοποίηση της κατάλληλης συνάρτησης κόστους, είδαμε τις βασικές στιγμές της εξέλιξής τους. Τα τελευταία χρόνια η δημοτικότητα των νευρωνικών δικτύων έχει ανέβει κατακόρυφα, καθώς μια νέα οικογένεια αλγορίθμων μηχανικής μάθησης αναπτύσσεται, μέθοδοι που συγκεντρωτικά ονομάζονται deep learning (βαθιά μηχανική μάθηση) και συνδέονται άμεσα με τα νευρωνικά δίκτυα.

Το σύγχρονο deep learning παρέχει ένα ισχυρό πλαίσιο επιβλεπόμενης (supervised) μάθησης. Προσθέτοντας περισσότερα layers και περισσότερα στοιχεία σε κάθε layer, ένα βαθύ δίκτυο (deep network) μπορεί να αναπαραστήσει όλο και πιο πολύπλοκες συναρτήσεις. Τα περισσότερα προβλήματα αντιστοίχισης ενός διανύσματος εισόδου σε ένα διάνυσμα εξόδου, τα οποία είναι εύκολα για τον άνθρωπο, μπορούν να επιλυθούν με deep learning, εφόσον εφαρμόσουμε αρκετά μεγάλα μοντέλα και έχουμε στη διάθεσή μας αρκετά δείγματα εκπαίδευσης.

Μπορούμε να πούμε, [8], ότι το deep learning ως κλάδος της Μηχανικής Μάθησης, εφαρμόζει αλγορίθμους για την επεξεργασία δεδομένων και τη μίμηση της νοητικής διαδικασίας ή την εξαγωγή γενικών κανόνων και εννοιών (abstractions) στους οποίους υπακούν τα δεδομένα.

Η εξαγωγή χαρακτηριστικών μέσω κάποιας μεθόδου είναι, επίσης, ένας τομέας του deep learning. Παράδειγμα σε αυτό, αποτελούν τα Convolutional Neural Networks όταν χρησιμοποιούνται στην όραση υπολογιστών, τα οποία μέσω διαδοχικών convolutional (συνελικτικών) και pooling layers εξάγουν χαρακτηριστικά της εικόνας χρήσιμα για να την ταξινομήσουν. Τα CNNs θα αναλυθούν σε βάθος στο πέμπτο κεφάλαιο της εργασίας.

Ένας συνοπτικός ορισμός βρίσκεται εδώ [9]: Το deep learning είναι μια νέα περιοχή έρευνας στη Μηχανική Μάθηση, που στόχο έχει να μεταφέρει τη Μηχανική Μάθηση πιο κοντά στον πραγματικό της στόχο, την Τεχνητή Νοημοσύνη. Το deep learning αφορά μάθηση πολλαπλών επιπέδων αναπαράστασης και γενικών κανόνων που βοηθούν στην κατανόηση δεδομένων όπως είναι οι εικόνες, ο ήχος και το κείμενο.

2.9 Η ανάγκη για Deep Networks

Στην παράγραφο 2.3 παρουσιάστηκε ένα απλό παράδειγμα νευρωνικού δικτύου με βαθιά αρχιτεκτονική, με πολλαπλά επίπεδα αναπαράστασης δηλαδή, απαραίτητα για τη σωστή ταξινόμηση προτύπων σε 2 μη-γραμμικά διαχωρίσιμες κλάσεις. Το παράδειγμα αυτό μπορεί να λειτουργήσει σαν σημείο εκκίνησης για να κατανοήσουμε τα deep networks. Το input layer περιγράφει κάθε πρότυπο σαν σημείο του χώρου εισόδου. Το 1ο hidden layer τοποθετεί το σημείο εισόδου σε μία από τις περιοχές χρησιμοποιώντας μια κωδικοποίηση από 1 και 0 στην έξοδο των αντίστοιχων νευρώνων. Αυτό, μπορεί να λογιστεί ως μια πιο αφαιρετική αναπαράσταση (abstract representation) των προτύπων εισόδου. Το 2ο hidden layer, βασισμένο στην πληροφορία που προέρχεται από το προηγούμενο layer, κωδικοποιεί μια πληροφορία που σχετίζεται με την κατηγορία. Η

αναπαράσταση των δεδομένων εμπεριέχει πλέον κάποιας μορφής σημασιολογική έννοια (semantic meaning). [3, σ 904]

Αυτή η ιεραρχικού τύπου αναπαράσταση των προτύπων εισόδου μιμείται τον τρόπο με τον οποίο ο εγκέφαλος "αντιλαμβάνεται" και "νιώθει" τον κόσμο που μας περιβάλλει. Αυτός είναι ο φυσικός μηχανισμός του μυαλού στον οποίο στηρίζεται η νοημοσύνη. Ο εγκέφαλος (των θηλαστικών) είναι οργανωμένος σε πολλά στρώματα νευρώνων και το κάθε στρώμα παρέχει μια διαφορετική αναπαράσταση του αισθήματος εισόδου. Με αυτόν τον τρόπο, διαφορετικά επίπεδα αφαιρετικής αντίληψης επιτυγχάνονται, μέσω της ιεραρχίας των μετασχηματισμών που επιτελούνται. Για παράδειγμα, στην πρωτεύουσα φάση του οπτικού μας συστήματος, αυτή η ιεραρχία εμπεριέχει ανίχνευση ακμών, πρωταρχικά σχήματα και όσο προχωράμε προς τα ανώτερα επίπεδα της ιεραρχίας, πιο πολύπλοκα σχήματα συμπεριλαμβάνονται ώσπου τελικά μια σημασιολογική έννοια γίνεται αντιληπτή, για παράδειγμα ένα αυτοκίνητο που κινείται ή ένας άνθρωπος που κάθεται. Όσον αφορά το οπτικό σύστημα, ο εγκέφαλός μας μπορεί να παρασταθεί σαν μια βαθιά αρχιτεκτονική που αποτελείται από 5 έως 10 layers. [10]

Το ερώτημα τώρα είναι κατά πόσο κάποιος μπορεί να επιτύχει παρόμοιες αναπαραστάσεις εισόδου-εξόδου χρησιμοποιώντας σχετικά απλά συναρτησιακά μοντέλα (όπως αυτά που εφαρμόζουν support vector machines) ή νευρωνικά δίκτυα με λιγότερα από τρία layers νευρώνων (ρηχά δίκτυα).

Η απάντηση στο πρώτο σκέλος είναι θετική, όσο η σχέση μεταξύ εισόδου και εξόδου είναι αρκετά απλή. Ωστόσο, για πιο πολύπλοκα προβλήματα, στα οποία πιο πολύπλοκες έννοιες πρέπει να γίνουν αντιληπτές, όπως στην αναγνώριση μιας σκηνής σε ένα βίντεο, μιας ομιλίας ή ενός κειμένου, η βαθύτερη λειτουργική εξάρτηση είναι αυξημένης πολυπλοκότητας και συνεπώς δεν μπορούμε να την εκφράσουμε αναλυτικά με απλό τρόπο.

Στο δεύτερο σκέλος, η απάντηση φαίνεται να σχετίζεται με το πόσο συμπαγής είναι η επιθυμητή αναπαράσταση (compactness of representation). Ένα νευρωνικό δίκτυο που περιγράφει τη λειτουργική εξάρτηση μεταξύ της εισόδου και της εξόδου, είναι συμπαγές αν αποτελείται από σχετικά μικρό αριθμό ελεύθερων παραμέτρων, οι οποίες ρυθμίζονται κατά τη φάση της εκπαίδευσης. Έτσι, για συγκεκριμένο αριθμό δειγμάτων εκπαίδευσης, αναμένουμε ότι περισσότερο συμπαγείς αναπαραστάσεις θα έχουν καλύτερη ικανότητα γενίκευσης.

Προκύπτει ότι χρησιμοποιώντας δίκτυα με περισσότερα layers, μπορούμε να επιτύχουμε πιο συμπαγείς αναπαραστάσεις της σχέσης εισόδου-εξόδου. Μολονότι δεν υπάρχουν ισχυρές θεωρητικές αποδείξεις του ισχυρισμού αυτού, από την πράξη φαίνεται ότι η ιεραρχικού τύπου αναπαράσταση των deep networks οδηγεί, με την κατάλληλη εκπαίδευση, σε ισχυρότερη ικανότητα γενίκευσης.

3. REGULARIZATION KAI DROPOUT

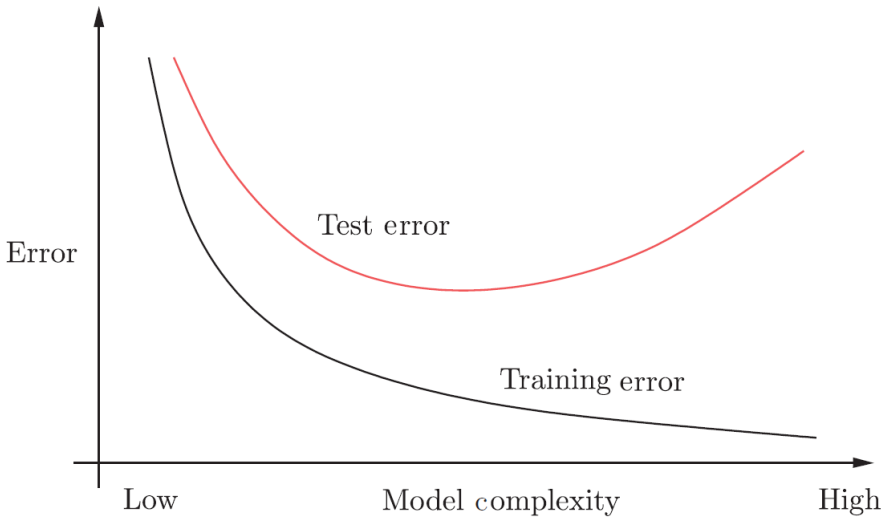
3.1 Το πρόβλημα του Overfitting

Τα περισσότερα προβλήματα στη μηχανική μάθηση ανήκουν στην κατηγορία των αντίστροφων προβλημάτων (inverse problems). Αυτός ο όρος συγκεντρώνει όλα τα προβλήματα στα οποία κάποιος πρέπει να συμπεράνει/προβλέψει/εκτιμήσει τις τιμές ενός μοντέλου από ένα σύνολο εκπαίδευσης παρατηρήσεων εισόδου/εξόδου. Με απλούστερα λόγια, στα αντίστροφα προβλήματα κάποιος πρέπει να ανιχνεύσει τις αιτίες που δημιούργησαν γνωστά αποτελέσματα, να αντιστρέψει δηλαδή τη σχέση αιτίας-αποτελέσματος. [3, σ 74]

Για την όσο το δυνατόν καλύτερη περιγραφή των διαθέσιμων δεδομένων, χρησιμοποιούμε πολύπλοκα μοντέλα, με την έννοια ότι το πλήθος των άγνωστων ελεύθερων παραμέτρων τους είναι μεγάλο σε σχέση με τον αριθμό των διαθέσιμων παρατηρήσεων. Αυτό είναι ταυτόσημο με εκείνο που στη μηχανική μάθηση ονομάζουμε overfitting, την υπερπροσαρμογή του μοντέλου στα δεδομένα εκπαίδευσης.

Ποιος είναι όμως ο λόγος που θέλουμε απαραιτήτως να αποφύγουμε αυτή την υπερπροσαρμογή; Το μοντέλο μας αναπτύσσεται με σκοπό να αντιμετωπίσει δεδομένα τα οποία δεν θα του είναι γνωστά. Αυτό σημαίνει πως θα πρέπει να έχει τη δυνατότητα γενίκευσης από τα δεδομένα εκπαίδευσης σε άλλα με τα οποία δεν έχει έρθει σε επαφή. Έτσι, γεννιέται, αρχικά, η ανάγκη της αξιολόγησης του μοντέλου μας σε δεδομένα που προέρχονται από έναν κόσμο άγνωστο σε αυτό. Δεν θα πρέπει να παραγνωρίσουμε το γεγονός ότι για το μοντέλο, όλος ο κόσμος είναι το σύνολο εκπαίδευσης που του παρέχουμε. Από εκεί οφείλει να εκπαιδευτεί ώστε να μπορεί να γενικεύσει σε άγνωστα δεδομένα.

Πέραν, λοιπόν, του συνόλου εκπαίδευσης (training set), χρησιμοποιούμε ένα σύνολο ελέγχου (test set) με παρατηρήσεις που δεν μετέχουν στην εκπαίδευση για να συμπεράνουμε την ικανότητα γενίκευσης του μοντέλου/εκτιμητή. Στην παρακάτω εικόνα φαίνεται μια τυπική απόδοση που αναμένεται στην πράξη.[3, σ 91]



Εικόνα 3.1: Test error και Training error συναρτήσει της πολυπλοκότητας του μοντέλου

Το σφάλμα που μετριέται στο training dataset παρουσιάζεται σε συνδυασμό με εκείνο που μετριέται στο test dataset καθώς η πολυπλοκότητα του μοντέλου αλλάζει. Αν κάποιος εφαρμόσει ένα μοντέλο με πολλές ελεύθερες παραμέτρους, σε σχέση με το πλήθος των δεδομένων εκπαίδευσης, το training error μπορεί ακόμα και να μηδενιστεί,

καθώς ένα τέλειο ταίριασμα (fit) στα δεδομένα μπορεί να επιτευχθεί. Η ικανότητα γενίκευσης του εκτιμητή όμως, σε άγνωστα δεδομένα, μειώνεται δραστικά. Από την άλλη, αν το μοντέλο είναι πολύ απλό, πάλι το test error παίρνει μεγάλες τιμές. Τι θα πρέπει, λοιπόν, να κάνουμε;

Έχοντας επιλέξει τον τύπο του εκτιμητή, θα επιδιώξουμε να έχουμε την πολυπλοκότητα εκείνη που αντιστοιχεί στο ελάχιστο της καμπύλης του test error. Στην πράξη, μπορούμε να χρησιμοποιήσουμε ένα validation set (σύνολο αξιολόγησης) για τη μέτρηση της ικανότητας γενίκευσης του μοντέλου και τη ρύθμιση κάποιων υπερ-παραμέτρων του, τη μεταβολή δηλαδή της πολυπλοκότητας του.

Φυσικά, στα παραπάνω θεωρήσαμε ότι έχουμε στη διάθεσή μας ένα σταθερό (περιορισμένο) training dataset. Σε περίπτωση που το πλήθος των δεδομένων εκπαίδευσης αυξηθεί δραστικά, αυτό συμβάλει καθοριστικά στο να μην υπερπροσαρμοστεί το μοντέλο μας σε αυτά. Διατηρούμε στη σκέψη μας την ιδεατή περίπτωση κατά την οποία διαθέτουμε όλα τα δυνατά δεδομένα για να κάνουμε μια εκτίμηση. Εκεί, δεν χρειαζόμαστε ούτε training set, ούτε test set. Έχουμε κατευθείαν την αναμενόμενη τιμή (expected value) της εξόδου μας για την είσοδο που δίνουμε, $\mathrm{E}[\boldsymbol{Y}|\boldsymbol{X}]$, η οποία είναι και η απάντηση στο πρόβλημά μας.

Μια ενδιαφέρουσα σχετική ιστορία θέλει τον κάτοχο του βραβείου Nobel, φυσικό, Enrico Fermi [11, Κεφ. 3] να ρωτήθηκε κάποτε την άποψή του για ένα μαθηματικό μοντέλο, που είχε προταθεί από μια ομάδα φοιτητών, για την επίλυση ενός σοβαρού φυσικού προβλήματος. Το μοντέλο είχε απόλυτη συμφωνία με τα πειραματικά δεδομένα, άλλα ο Fermi ήταν σκεπτικός. Όταν ρώτησε πόσες ήταν οι ελεύθερες παράμετροι που μπορούσαν να ρυθμιστούν, η απάντηση ήταν τέσσερις. Ο Fermi τότε αποκρίθηκε: "Θυμάμαι τον φίλο μου Johnny von Neumann που συνήθιζε να λέει, με τέσσερις παραμέτρους μπορώ να ταιριάξω έναν ελέφαντα στα δεδομένα και με πέντε μπορώ να τον κάνω να κουνάει την προβοσκίδα του". [12]

3.2 Regularization για Deep Learning

Αναπτύχθηκε παραπάνω ότι ένα κεντρικό πρόβλημα στη μηχανική μάθηση είναι η ανάπτυξη ενός αλγορίθμου που όχι μόνο αποδίδει καλά στα δεδομένα εκπαίδευσης, άλλα και σε άγνωστα σε αυτό. Πολλές στρατηγικές που χρησιμοποιούνται στο machine learning στοχεύουν στη μείωση του test error, πολλές φορές με κόστος την αύξηση του training error. Οι στρατηγικές αυτές συγκεντρώνονται όλες μαζί υπό τον όρο regularization (μέθοδοι εξομάλυνσης). [7, σ 228]

Με τον όρο regularization ορίζουμε κάθε μετατροπή που κάνουμε σε έναν αλγόριθμο μάθησης με σκοπό να μειώσουμε το σφάλμα στη γενικευτική του ικανότητα και όχι το σφάλμα του στο σύνολο εκπαίδευσης. Υπάρχουν διάφορες μέθοδοι εξομάλυνσης. Κάποιες προσθέτουν περιορισμούς στο μοντέλο μηχανικής μάθησης, όπως για παράδειγμα τη δέσμευση των παραμέτρων του σε συγκεκριμένες τιμές. Άλλες προσθέτουν επιπλέον όρους στην αντικειμενική συνάρτηση, κάτι που μπορεί να θεωρηθεί σαν μαλακός (soft) περιορισμός στις τιμές των παραμέτρων. Αν επιλεχθούν προσεκτικά, αυτοί οι επιπλέον περιορισμοί μπορούν να οδηγήσουν σε βελτίωση της απόδοσης στο test set.

Στο πλαίσιο του deep learning, οι περισσότερες μέθοδοι regularization βασίζονται στην εξομάλυνση του εκτιμητή (estimator). Το regularization ενός εκτιμητή αναφέρεται σε αύξηση του bias με αντίκρισμα τη μείωση του variance. Καλό είναι εδώ να αναφερθεί η ανάλυση του σφάλματος ενός estimator σε bias και variance.

Ένα λογικό κριτήριο για τη μέτρηση της απόδοσης ενός εκτιμητή f είναι η μέση τετραγωνική απόκλισή του από το βέλτιστο εκτιμητή, [3, σ 78],

$$\mathrm{E}_D[(f(\boldsymbol{x}; D) - \mathrm{E}[\mathrm{y}|\boldsymbol{x}])^2]$$

όπου D ένα συγκεκριμένο training set. Η μέση τιμή αναφέρεται σε όλα τα πιθανά διαφορετικά training sets, καθώς κάθε ένα από αυτά οδηγεί σε διαφορετικό εκτιμητή. Σημειώνεται ότι ακόμα και ο βέλτιστος εκτιμητής περιέχει ένα irreducible error, που σχετίζεται με τη φύση των δεδομένων μας ή/και την επιλογή του μοντέλου που έχουμε κάνει. Επιστρέφουμε στο σφάλμα του εκτιμητή το οποίο με ανάλυση προκύπτει,

$$\mathrm{E}_D[(f(\boldsymbol{x}; D) - \mathrm{E}[\mathrm{y}|\boldsymbol{x}])^2] = \mathrm{E}_D[(f(\boldsymbol{x}; D) - \mathrm{E}_D[f(\boldsymbol{x}; D)])^2] + (\mathrm{E}_D[f(\boldsymbol{x}; D)] - \mathrm{E}[\mathrm{y}|\boldsymbol{x}])^2.$$

Αλλάζοντας λοιπόν από ένα training set σε άλλο, η μέση τετραγωνική απόκλιση από τον βέλτιστο εκτιμητή περιλαμβάνει δύο όρους. Ο πρώτος οφείλεται στο variance (διακύμανση) του εκτιμητή γύρω από τη δική του μέση τιμή (στο σύνολο των εκτιμητών που προκύπτουν από τα διαφορετικά training sets), ενώ ο δεύτερος όρος στη διαφορά της μέσης τιμής του εκτιμητή από τη βέλτιστη εκτίμηση, δηλαδή το bias. Από την εμπειρία προκύπτει ότι κάποιος δεν μπορεί να ελαττώσει ταυτόχρονα και τους δύο όρους. Το παραπάνω είναι γνωστό ως bias-variance δίλημμα ή bias-variance tradeoff.

Για συγκεκριμένο πλήθος, παρατηρήσεων εκπαίδευσης, N, στα datasets D, η προσπάθεια για ελαχιστοποίηση του variance όρου θα οδηγήσει σε αύξηση του bias όρου και το αντίστροφο. Αυτό συμβαίνει γιατί, για να μειώσει κάποιος το bias, θα πρέπει να αυξήσει την πολυπλοκότητα του εκτιμητή (περισσότερες ελεύθερες παραμέτρους), πράγμα που οδηγεί στην αύξηση του variance του εκτιμητή καθώς μεταβάλλουμε τα training sets. Πρόκειται ουσιαστικά για ακόμα μία εκδοχή του overfitting. Ο μόνος τρόπος να μειώσουμε ταυτόχρονα και τους δύο όρους, είναι να αυξήσουμε το πλήθος των δεδομένων εκπαίδευσης, N, και παράλληλα να αυξήσουμε την πολυπλοκότητα του μοντέλου με προσοχή για να πετύχουμε τον σκοπό μας.

Στην πράξη, [7, σ 229] ακόμα και ένα υπερβολικά πολύπλοκο μοντέλο δεν είναι απαραίτητο ότι θα περιέχει την πραγματική διαδικασία δημιουργίας των δεδομένων, ή έστω μια καλή προσέγγιση της διαδικασίας αυτής. Για την ακρίβεια, δεν έχουμε σχεδόν ποτέ πρόσβαση στην πραγματική διαδικασία γέννησης των δεδομένων και συνεπώς δεν μπορούμε να ξέρουμε αν αυτή εμπεριέχεται ή όχι στο μοντέλο μας.

Επιπρόσθετα, οι περισσότερες εφαρμογές των deep learning αλγορίθμων είναι σε προβλήματα στα οποία η πραγματική διαδικασία γέννησης των δεδομένων βρίσκεται σχεδόν ολοκληρωτικά έξω από το μοντέλο. Οι αλγόριθμοι βαθιάς μηχανικής μάθησης εφαρμόζονται συνήθως σε ιδιαιτέρως πολύπλοκα προβλήματα όπως σε εικόνες, ηχητικές ακολουθίες και κείμενο, στα οποία η πραγματική γενεσιουργός διαδικασία συμπεριλαμβάνει σχεδόν όλον τον κόσμο. Αυτό σημαίνει ότι ο έλεγχος της πολυπλοκότητας του μοντέλου δεν είναι ένα απλό ζήτημα προσδιορισμού του κατάλληλου μεγέθους του, με τον κατάλληλο αριθμό παραμέτρων. Αυτό που κάνουμε, στα πλαίσια του deep learning, είναι να προσδιορίζουμε ένα μεγάλο, πολύπλοκο μοντέλο στο οποίο παράλληλα εφαρμόζουμε κατάλληλο regularization. Είναι, λοιπόν, κομβικής σημασίας η διαδικασία της εξομάλυνσης στο deep learning.

3.3 Εξομάλυνση με Ποινές στη Νόρμα των Παραμέτρων

Το regularization χρησιμοποιείται για δεκαετίες, πολύ πριν την ανάπτυξη του deep learning. Γραμμικά μοντέλα όπως η γραμμική παλινδρόμηση (linear regression) και η

λογιστική παλινδρόμηση (logistic regression) επιτρέπουν απλό, ευθύ και αποτελεσματικό regularization. [7, σ 230]

Παραδοσιακά, πολλές μέθοδοι εξομάλυνσης βασίζονται στην αποδοκιμασία των μεγάλων τιμών των παραμέτρων, προσθέτοντας μια παράμετρο ποινής νόρμας $\Omega(\boldsymbol{\theta})$ στη συνάρτηση κόστους J. Η εξομαλυμένη συνάρτηση κόστους είναι τότε,

$$\tilde{J}(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) = J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y}) + a\Omega(\boldsymbol{\theta})$$

όπου $\alpha \in [0, \infty)$ μια υπερπαράμετρος που καθορίζει τη συμβολή του όρου ποινής Ω σε σχέση με την αντικειμενική συνάρτηση J. Θέτοντας το α στην τιμή 0, δεν έχουμε καθόλου regularization ενώ μεγαλύτερες τιμές του α οδηγούν σε μεγαλύτερη εξομάλυνση.

Όταν ο εκπαιδευόμενος αλγόριθμος ελαχιστοποιεί τη συνολική συνάρτηση κόστους $\tilde{J}$, θα επιδιώξει τη μείωση τόσο της αρχικής συνάρτησης J για τα δεδομένα εκπαίδευσης όσο και ενός επιλεγμένου μέτρου του μεγέθους των παραμέτρων $\boldsymbol{\theta}$ ή ενός υποσυνόλου των παραμέτρων αυτών. Διαφορετικές επιλογές του όρου ποινής (penalty term) Ω, θα οδηγήσουν σε διαφορετική διαμόρφωση του μοντέλου.

Πριν την παρουσίαση των διαφορετικών συμπεριφορών που προκύπτουν με τη χρησιμοποίηση διαφορετικών μέτρων του μεγέθους των παραμέτρων, σημειώνουμε ότι για τα νευρωνικά δίκτυα, χρησιμοποιούμε κατά κανόνα έναν όρο ποινής νόρμας που τιμωρεί μόνο τα συναπτικά βάρη σε κάθε layer και αφήνει τα biases χωρίς regularization. Οι bias όροι χρειάζονται συνήθως λιγότερα δεδομένα για να ρυθμιστούν σωστά σε σχέση με τα συναπτικά βάρη.

Κάθε συναπτικό βάρος προσδιορίζει πως αλληλεπιδρούν δύο μεταβλητές. Η σωστή ρύθμισή του προϋποθέτει την παρατήρηση των δύο αυτών μεταβλητών σε μια πληθώρα συνθηκών. Κάθε bias, όμως, ελέγχει μία μόνο μεταβλητή. Αυτό σημαίνει ότι δεν εισάγουμε πολύ variance στο μοντέλο μας αφήνοντας τους όρους bias χωρίς εξομάλυνση. Επιπλέον, η εφαρμογή regularization στα biases μπορεί να οδηγήσει σε underfitting του αλγορίθμου ως προς τις παραμέτρους αυτές, κάτι που δεν επιθυμούμε επίσης. Συμβολίζουμε με $\boldsymbol{w}$ τα συναπτικά βάρη, στα οποία εφαρμόζουμε regularization, και με $\boldsymbol{\theta}$ το σύνολο των παραμέτρων, το οποίο περιέχει και τους όρους τους οποίους δεν εξομαλύνουμε.

Όσον αφορά τον συντελεστή του όρου ποινής νόρμας α, είναι επιθυμητή η χρησιμοποίηση διαφορετικής τιμής για κάθε layer. Επειδή αυτό μπορεί να έχει κόστος στην εκτέλεση του αλγορίθμου, είναι αποδεκτός ο ορισμός ενός ενιαίου συντελεστή του όρου ποινής νόρμας για όλα τα layers, ώστε να μειωθεί ο χώρος αναζήτησης.

3.4 L^2 και L^1 Regularization

Όπως προαναφέρθηκε, το regularization είναι ένα μαθηματικό εργαλείο για να εισάγουμε a priori πληροφορία στη λύση μας, η οποία προκύπτει ως έξοδος ενός προβλήματος βελτιστοποίησης. Η εξομάλυνση προτάθηκε αρχικά από τον σπουδαίο Ρώσο μαθηματικό Andrey Nikolayevich Tychonoff για την επίλυση ολοκληρωτικών εξισώσεων. Η μέθοδος αυτή αναφέρεται κάποιες φορές ως Tychonoff-Phillips regularization προς τιμήν και του David Phillips, που την ανέπτυξε ανεξάρτητα την ίδια περίοδο.

L^2 Regularization

Στο πλαίσιο που έχουμε αναφέρει, επιθυμούμε να διατηρήσουμε μικρό το μέγεθος της νόρμας των παραμέτρων παράλληλα με την ελαχιστοποίηση της αντικειμενικής συνάρτησης,

$$\text{ελαχιστοποίηση: } J(\boldsymbol{\theta}; \boldsymbol{X}, \boldsymbol{y})$$

$$\text{υπό τον περιορισμό: } \|\boldsymbol{\theta}\|_2^2 \leq \rho$$

όπου $\|.\|_2$ η ευκλείδεια νόρμα ενός διανύσματος. Με αυτόν τον τρόπο, δεν αφήνουμε το μοντέλο μας εντελώς ελεύθερο να προσδιορίσει κάποια λύση, αλλά περιορίζουμε τον χώρο στον οποίο θα την αναζητήσει. Προφανώς, διαφορετικές τιμές του ρ, θα οδηγήσουν σε διαφορετικά επίπεδα συστολής της νόρμας των παραμέτρων. Η βέλτιστη τιμή του ρ δεν μπορεί να εξαχθεί με αναλυτικό τρόπο και συνεπώς κάποιος θα πρέπει πειραματικά να την προσδιορίσει. [3, σ 72]

Για να επιτύχουμε το παραπάνω, προσθέτουμε τον regularization όρο $\Omega(\boldsymbol{\theta}) = \frac{1}{2}\|\boldsymbol{w}\|_2^2$ στη συνάρτηση κόστους J. Σημειώνεται ότι τα biases δεν συμμετέχουν στην εξομάλυνση, οπότε το $\boldsymbol{\theta}$ είναι απλά $\boldsymbol{w}$. Τότε, έχουμε τη συνάρτηση κόστους με regularization,

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = \frac{a}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Για συγκεκριμένες τιμές των $a \geq 0$ και ρ, τα δύο παραπάνω προβλήματα είναι ταυτόσημα. Συχνά η εξομάλυνση αυτή αναφέρεται ως ridge regression ή Tychonoff regularization ή απλά weight decay. Μπορούμε να αποκτήσουμε μια καλύτερη κατανόηση της L^2 εξομάλυνσης μελετώντας το gradient της συνάρτησης κόστους με regularization,

$$\nabla_{\boldsymbol{w}}\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = a\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Σε ένα μοναδικό βήμα, η ανανέωση των βαρών θα είναι,

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \epsilon(a\boldsymbol{w} + \nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}))$$

Γραμμένο διαφορετικά,

$$\boldsymbol{w} \leftarrow (1 - \epsilon\alpha)\boldsymbol{w} - \epsilon\nabla_{\boldsymbol{w}}J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Από το παραπάνω γίνεται εμφανές ότι η προσθήκη του όρου εξομάλυνσης, μετέτρεψε τον κανόνα εκπαίδευσης έτσι ώστε να συρρικνώνει κατά μία σταθερά $(\epsilon\alpha)$ το διάνυσμα των βαρών σε κάθε βήμα, πριν το συνηθισμένο gradient update. [7, σ 231]

L^1 Regularization

Αν και η $\boldsymbol{L^2}$ εξομάλυνση είναι η πιο συχνά χρησιμοποιούμενη για την απομείωση των βαρών, υπάρχουν και άλλες μέθοδοι που εφαρμόζονται ως ποινές στο μέγεθος των παραμέτρων.

Μία από αυτές τις μεθόδους είναι η $\boldsymbol{L^1}$ regularization, για την οποία χρησιμοποιούμε τον όρο,

$$\Omega(\boldsymbol{\theta}) = \|\boldsymbol{w}\|_1 = \sum_i |w_i|$$

Πρόκειται, δηλαδή, για το άθροισμα των απολύτων τιμών όλων των παραμέτρων. Με την προσθήκη του όρου αυτού, παίρνουμε την αντικειμενική συνάρτηση με regularization,

$$\tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = a\|\boldsymbol{w}\|_1 + J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y})$$

Η αντίστοιχη παράγωγος είναι τώρα,

$$\nabla_{\boldsymbol{w}} \tilde{J}(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) = a\,\mathrm{sign}(\boldsymbol{w}) + \nabla_{\boldsymbol{w}} J(\boldsymbol{w}; \boldsymbol{X}, \boldsymbol{y}) \ ,$$

όπου $\mathrm{sign}(\boldsymbol{w})$ είναι απλά το πρόσημο του καθενός βάρους w κατά το update. Μελετώντας την παραπάνω εξίσωση, βλέπουμε ότι η επίδραση της $\boldsymbol{L^1}$ εξομάλυνσης είναι αρκετά διαφορετική από εκείνη της $\boldsymbol{L^2}$. Συγκεκριμένα, η συνεισφορά του regularization όρου στη συνάρτηση κόστους δεν οδηγεί εδώ σε γραμμική συρρίκνωση του κάθε βάρους w_i ανάλογα με το μέγεθός του. Αντί αυτού, είναι ένας σταθερός όρος με πρόσημο $\mathrm{sign}(w_i)$ που προστίθεται κατά την ενημέρωση των παραμέτρων. [7, σ 235]

Για να ειπωθεί πιο καθαρά, και στις δύο περιπτώσεις η συνέπεια της εξομάλυνσης είναι η συρρίκνωση του μεγέθους των παραμέτρων. Ο τρόπος, όμως, που επιτυγχάνεται αυτό είναι διαφορετικός. Στην $\boldsymbol{L^1}$ εξομάλυνση τα βάρη συρρικνώνονται κατά μία σταθερή τιμή γύρω από το 0, ενώ στην $\boldsymbol{L^2}$ κατά μία ποσότητα ανάλογη της νόρμας τους. Έτσι, σε περίπτωση που ένα βάρος έχει μεγάλο πλάτος $|w|$, η $\boldsymbol{L^1}$ εξομάλυνση θα συρρικνώσει το μέγεθός του πολύ λιγότερο από την $\boldsymbol{L^2}$, και το αντίθετο στην περίπτωση που το βάρος έχει μικρό πλάτος.

Το τελικό αποτέλεσμα είναι ότι η $\boldsymbol{L^1}$ εξομάλυνση τείνει να συγκεντρώνει το διάνυσμα των βαρών του δικτύου σε έναν σχετικά μικρό αριθμό από συνδέσεις υψηλής σημαντικότητας, οδηγώντας παράλληλα τα βάρη των υπολοίπων κοντά στο μηδέν. [11, Κεφ. 3]

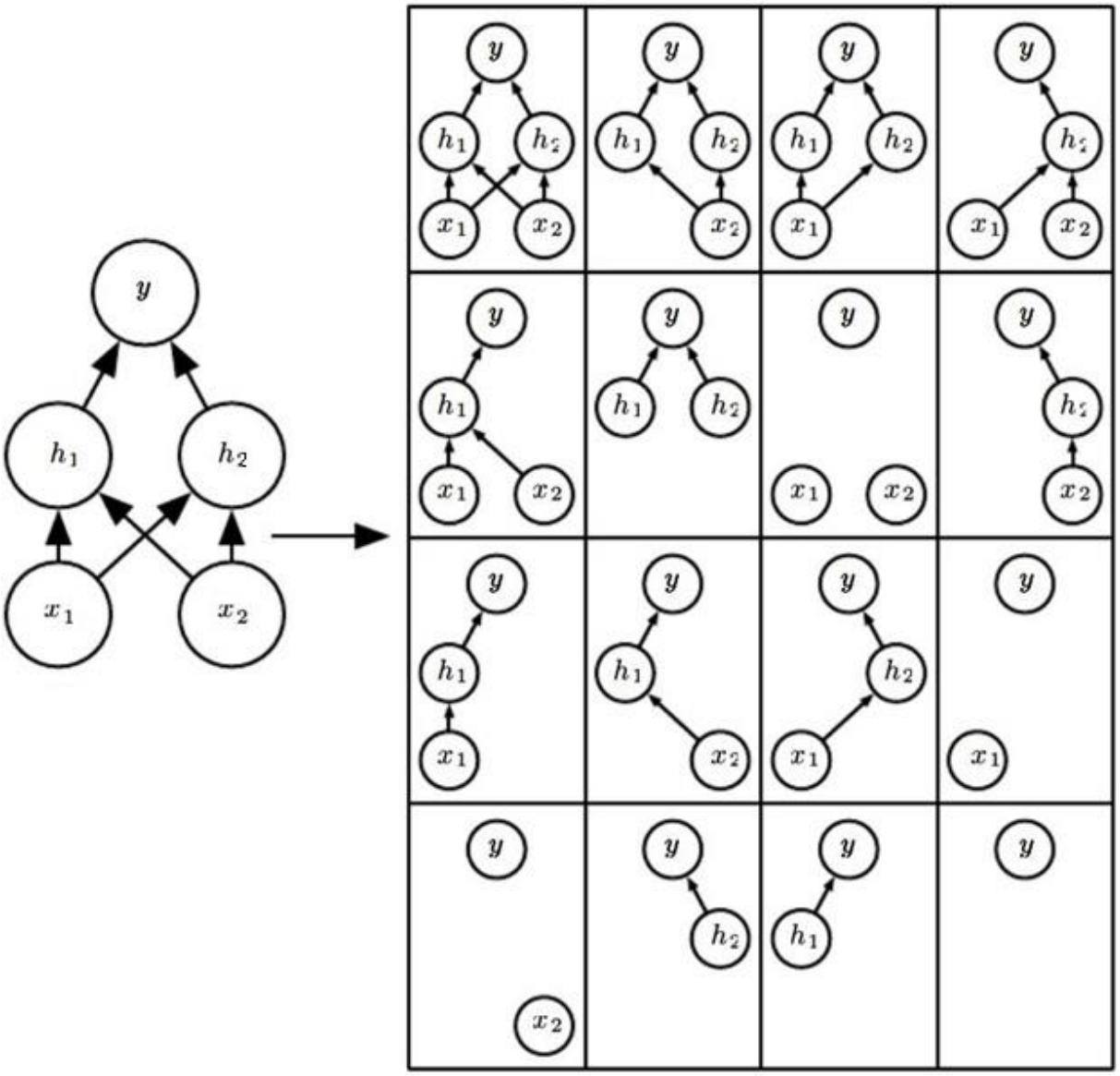
3.5 Dropout

3.5.1 Εισαγωγή

Το Dropout παρέχει μια ισχυρή και υπολογιστικά όχι ακριβή μέθοδο για την εξομάλυνση μιας ευρείας οικογένειας προβλημάτων. Σε μια πρώτη προσέγγιση, το dropout μπορεί να περιγράφει σαν μια πρακτική μέθοδος σύνθεσης πολλών μεγάλων νευρωνικών δικτύων. Κάτι τέτοιο φαίνεται μη λειτουργικό όταν κάθε μοντέλο είναι ένα πολύ μεγάλο νευρωνικό δίκτυο που η εκπαίδευση και η αξιολόγησή του είναι ακριβή τόσο υπολογιστικά όσο και σε απαραίτητη μνήμη. Είναι σύνηθες να χρησιμοποιούνται σύνολα από πέντε έως δέκα νευρωνικά δίκτυα για τον ίδιο σκοπό, αλλά περισσότερα από τόσα καθιστούν σύντομα την υλοποίηση ανέφικτη. Το dropout παρέχει μία εφικτή - όχι ακριβή - προσέγγιση ώστε να εκπαιδεύσουμε και να χρησιμοποιήσουμε ένα σύνολο εκθετικού πλήθους νευρωνικών δικτύων. [7, σ 258]

Συγκεκριμένα, με το dropout εκπαιδεύουμε το σύνολο που αποτελείται από όλα τα υπο-δίκτυα που μπορούν να σχηματιστούν αφαιρώντας νευρώνες, εκτός από αυτούς του

στρώματος εξόδου, από το συνολικό νευρωνικό δίκτυο, όπως φαίνεται και στην εικόνα παρακάτω:



Εικόνα 3.2: Αρχικό δίκτυο και όλα τα δυνατά υπο-δίκτυα διατηρώντας το νευρώνα εξόδου

Στα σύγχρονα νευρωνικά δίκτυα, που βασίζονται σε μια σειρά διαδοχικών μετασχηματισμών και μη-γραμμικοτήτων, μπορούμε άνετα να αφαιρέσουμε κάποιον νευρώνα πολλαπλασιάζοντας την έξοδό του με το μηδέν. Η διαδικασία αυτή θα πρέπει ελαφρώς να διαφοροποιηθεί σε μοντέλα όπως τα νευρωνικά με συναρτήσεις ακτινικής βάσης, τα οποία λαμβάνουν υπόψη τη διαφορά της κατάστασης του νευρώνα από μια τιμή αναφοράς.

Το dropout προσομοιάζει αρκετά την εκπαίδευση ενός συνόλου μοντέλων για το ίδιο πρόβλημα (bagging training) και στοχεύει στην όσο το δυνατόν καλύτερη προσέγγιση της διαδικασίας αυτής, χρησιμοποιώντας ένα εκθετικά μεγάλο αριθμό δικτύων. Επισημαίνουμε ότι για την εκμάθηση ενός συνόλου μοντέλων (bagging), ορίζουμε k διαφορετικά μοντέλα, φτιάχνουμε k διαφορετικά datasets με δειγματοληψία στο training set και εκπαιδεύουμε το μοντέλο i στο dataset i.

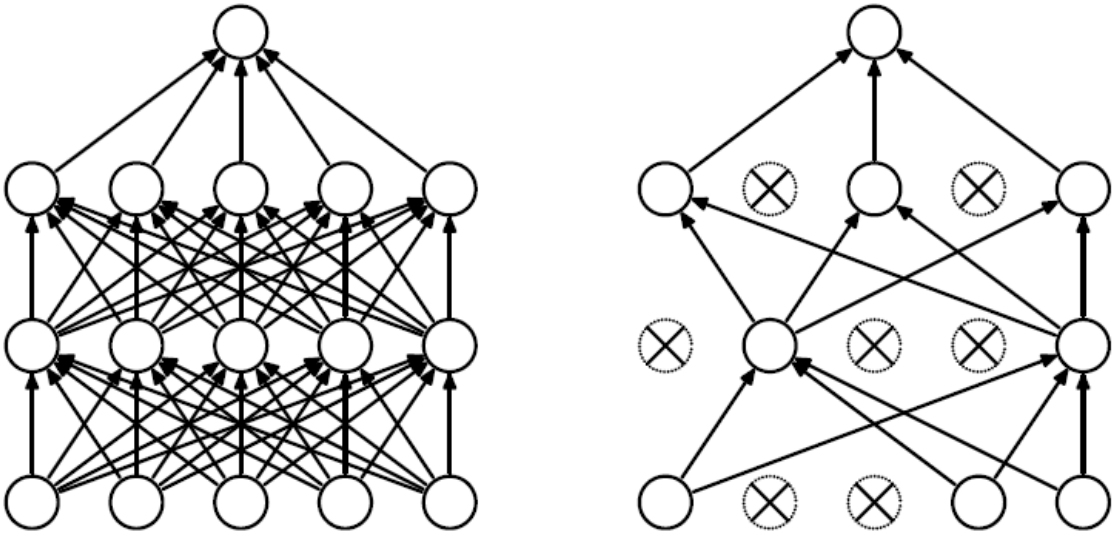
Η εκπαίδευση με dropout είναι σε κάποια σημεία διαφορετική. Πιο συγκεκριμένα, χρησιμοποιούμε έναν αλγόριθμο που δέχεται μικρά σύνολα παρατηρήσεων

(minibatches) και κάνει μικρά βήματα, όπως ο stochastic gradient descent. Κάθε φορά που παρουσιάζουμε ένα παράδειγμα από το minibatch, εφαρμόζουμε μια τυχαία δυαδική μάσκα στο δίκτυό μας, η οποία καθορίζει τη συμμετοχή ή μη κάθε νευρώνα - εισόδου ή κρυφού στρώματος - στο δίκτυο. Η τιμή της μάσκας για τον κάθε νευρώνα είναι ανεξάρτητη των υπολοίπων. Η πιθανότητα η τιμή της μάσκας σε κάποιον νευρώνα να έχει την τιμή 1, δηλαδή ο νευρώνας να περιλαμβάνεται στο δίκτυο, είναι μια υπερ-παράμετρος (hyperparameter) η οποία καθορίζεται πριν ξεκινήσει η φάση της εκπαίδευσης. Δεν είναι συνάρτηση της τρέχουσας τιμής των παραμέτρων του μοντέλου, ούτε και του παραδείγματος εισόδου. Τυπικές τιμές της πιθανότητας συμμετοχής ενός νευρώνα στο δίκτυο είναι 0.8 για εκείνους του στρώματος εισόδου και 0.5 για τους νευρώνες των κρυφών στρωμάτων. Εφαρμόζουμε back-propagation, ως συνήθως, και ενημερώνουμε τις τιμές των παραμέτρων όπως σε κάθε νευρωνικό δίκτυο.

Έτσι, αν και έχουν αρκετή συνάφεια, το dropout δεν είναι ακριβώς το ίδιο με την εκπαίδευση πολλών μοντέλων (bagging training) για το ίδιο πρόβλημα. Στην περίπτωση εκείνη, τα μοντέλα είναι ανεξάρτητα. Στην περίπτωση του dropout, τα μοντέλα μοιράζονται (share) παραμέτρους, με το κάθε μοντέλο να κληρονομεί (inherit) ένα διαφορετικό υποσύνολο παραμέτρων από το γονικό (parent) νευρωνικό δίκτυο. Αυτή η κοινή χρήση παραμέτρων καθιστά εφικτή τη δειγματοληψία ενός εκθετικού πλήθους μοντέλων με πεπερασμένη μνήμη. Στην περίπτωση του bagging, το κάθε μοντέλο εκπαιδεύεται να συγκλίνει στο σύνολο εκπαίδευσης. Στο dropout, τα περισσότερα μοντέλα δεν εκπαιδεύονται αυτούσια καθώς κατά κανόνα το μοντέλο είναι τόσο μεγάλο που είναι αδύνατη η δειγματοληψία όλων των πιθανών υποδικτύων σε πεπερασμένο χρόνο. Αντί για αυτό, ένα μικρό κλάσμα των πιθανών υποδικτύων εκπαιδεύεται, και η κοινή χρήση των παραμέτρων οδηγεί στο να ρυθμιστούν καλά και τα υπόλοιπα υποδίκτυα. [7, σ 259]

3.5.2 Αναλυτική περιγραφή

Για να το δούμε περισσότερο αναλυτικά, η εφαρμογή του dropout σε ένα νευρωνικό δίκτυο οδηγεί στη δειγματοληψία ενός απομειωμένου (thinned) δικτύου από αυτό [13][12]. Το απομειωμένο δίκτυο αποτελείται από τους νευρώνες εκείνους που επιβίωσαν του dropout, όπως φαίνεται στην εικόνα:



Εικόνα 3.3: Αριστερά το συνολικό δίκτυο και δεξιά το δίκτυο μετά την εφαρμογή dropout

Κάθε δίκτυο με n νευρώνες μπορούμε να το δούμε σαν ένα σύνολο από 2^n πιθανά υποδίκτυα. Αφού τα υποδίκτυα αυτά μοιράζονται τα βάρη, ο συνολικός αριθμός των

παραμέτρων παραμένει $O(n^2)$ ή μικρότερος. Για κάθε παρουσίαση σε κάθε φάση εκπαίδευσης, ένα νέο υποδίκτυο δειγματολειπτείται και εκπαιδεύεται. Άρα, η εκπαίδευση ενός νευρωνικού δικτύου με dropout καταλήγει στην εκπαίδευση ενός συνόλου 2^n υποδικτύων τα οποία μοιράζονται τα βάρη τους, και το καθένα εκπαιδεύεται πολύ σπάνια, μπορεί και καθόλου. Κατά τον έλεγχο του μοντέλου (test time), δεν είναι δυνατόν να πάρουμε τον μέσο όρο των προβλέψεων από ένα εκθετικό αριθμό απομειωμένων δικτύων. Εντούτοις, μια πολύ απλή προσέγγιση δουλεύει καλά στην πράξη. Τα βάρη του δικτύου κανονικοποιούνται ανάλογα με την πιθανότητα επιβίωσης που είχαν οι νευρώνες τους κατά την εκπαίδευση. Έτσι, αν ένας νευρώνας επιβίωνε με πιθανότητα p κατά την εκπαίδευση, τα βάρη που ξεκινούν από αυτόν πολλαπλασιάζονται με p κατά τη φάση του ελέγχου του μοντέλου.

Αυτό διαβεβαιώνει ότι για κάθε νευρώνα, η αναμενόμενη έξοδος (που ακολουθεί την κατανομή που χρησιμοποιήθηκε για το «άφημα» νευρώνων κατά την εκπαίδευση) ταυτίζεται με την πραγματική έξοδο στη φάση του ελέγχου.

Με την κανονικοποίηση αυτή, τα 2^n δίκτυα συντίθενται σε ένα ενιαίο δίκτυο για να χρησιμοποιηθεί στον έλεγχο του μοντέλου. Έχει παρατηρηθεί πειραματικά, και φαίνεται στα επόμενα κεφάλαια της παρούσας εργασίας, ότι η εκπαίδευση ενός νευρωνικού δικτύου με dropout οδηγεί σε σημαντική μείωση του σφάλματος γενίκευσης (generalization error) σε μια πληθώρα προβλημάτων και με τη χρήση διαφόρων αλγορίθμων για την αντιμετώπισή τους. [13]

3.5.3 Σύνοψη

Το dropout είναι μια μέθοδος βελτίωσης της απόδοσης των νευρωνικών δικτύων μειώνοντας το overfitting, την υπερπροσαρμογή του δηλαδή στα δεδομένα εκπαίδευσης. Η χρήση του backpropagation κατά την εκμάθηση, αναπτύσσει ελαφριές αλληλο-προσαρμογές μεταξύ των νευρώνων οι οποίες δουλεύουν καλά για το σύνολο εκπαίδευσης, δεν παρέχουν όμως δυνατότητα γενίκευσης σε άγνωστα δεδομένα. Το τυχαίο dropout σπάει τις αλληλο-προσαρμογές αυτές καθιστώντας την παρουσία κάθε νευρώνα (εκτός εκείνων του στρώματος εξόδου) μη εξασφαλισμένη. Η μέθοδος αυτή έχει συμβάλει στην αύξηση της απόδοσης των νευρωνικών δικτύων σε μια πληθώρα εφαρμογών από την ταξινόμηση αντικειμένων και ψηφίων μέχρι την αναγνώριση ομιλίας, την ταξινόμηση κειμένου και την ανάλυση βιολογικών δεδομένων.

Στην παρούσα εργασία θα δούμε την εφαρμογή του dropout σε διαφορετικά δίκτυα, Feedforward και Convolutional, για την αναγνώριση ψηφίων γραμμένων με το χέρι καθώς επίσης τη χρήση του στην εκπαίδευση ενός Recurrent Neural Network για μοντελοποίηση γλώσσας.

4. TENSORFLOW ΚΑΙ DEEP FEEDFORWARD NN ΜΕ DROPOUT

4.1 Εισαγωγή στο TensorFlow

Ενώ οι μαθηματικές έννοιες πίσω από το deep learning υπάρχουν εδώ και δεκαετίες, οι βιβλιοθήκες για τον προγραμματισμό και την ανάπτυξη αυτών των μοντέλων είναι διαθέσιμες εδώ και μόλις κάποια χρόνια. Δυστυχώς, οι περισσότερες από αυτές τις βιβλιοθήκες εμπεριέχουν ένα μεγάλο συμβιβασμό ανάμεσα στην ευελιξία της ανάπτυξης και στην παραγωγική αξία αυτής. Ευέλικτες βιβλιοθήκες είναι εξαιρετικές για την έρευνα νέων αρχιτεκτονικών μοντέλων, αλλά είναι συχνά πολύ αργές ή ανίκανες να χρησιμοποιηθούν στην παραγωγή. Απ' την άλλη, γρήγορες και αποδοτικές βιβλιοθήκες που μπορούν να φιλοξενούνται σε διανεμημένα συστήματα (distributed hardware) είναι διαθέσιμες, αλλά ειδικεύονται συνήθως σε συγκεκριμένους τύπους νευρωνικών δικτύων και δεν είναι κατάλληλες για την έρευνα νέων μοντέλων.

Αυτό δημιουργεί στους μηχανικούς ένα δίλημμα: θα πρέπει να προσπαθήσουν να κάνουν έρευνα με μη ευέλικτες βιβλιοθήκες ώστε να μην χρειάζεται να επανα-υλοποιήσουν κώδικα ή είναι προτιμότερο να χρησιμοποιήσουν διαφορετική βιβλιοθήκη για την έρευνα απ' ό,τι για την παραγωγή;

Αν επιλέξουμε το πρώτο, πιθανότατα δεν θα μπορούμε να ελέγχουμε διαφορετικά νευρωνικά δίκτυα ενώ αν επιλέξουμε το δεύτερο θα πρέπει να συντηρούμε κώδικα που μάλλον θα έχει εντελώς διαφορετικά APIs. Το TensorFlow στοχεύει στο να επιλύσει το παραπάνω δίλημμα. [14]

Το TensorFlow είναι μια ανοιχτού λογισμικού βιβλιοθήκη για αριθμητικούς υπολογισμούς η οποία χρησιμοποιεί γράφους ροής δεδομένων (data flow graphs). Οι κόμβοι του γράφου αναπαριστούν μαθηματικές λειτουργίες, ενώ οι ακμές του αναπαριστούν τους πολυδιάστατους πίνακες των δεδομένων (tensors) που συνδέονται σε αυτές. Η ευέλικτη αρχιτεκτονική επιτρέπει την υλοποίηση των υπολογισμών σε έναν ή παραπάνω επεξεργαστές (CPUs) ή κάρτες γραφικών (GPUs) σε ένα desktop, server, ή μία φορητή συσκευή με τη χρήση ενός μοναδικού API. Το TensorFlow αναπτύχθηκε αρχικά από ερευνητές και μηχανικούς που εργάζονταν στην Google Brain Team για την έρευνα πάνω στη βαθιά μηχανική μάθηση, αλλά ως σύστημα είναι αρκετά γενικό ώστε να εφαρμοστεί και σε άλλα πεδία. [15]

Άλλες αντίστοιχες δημοφιλείς βιβλιοθήκες για deep learning είναι η Torch, που είναι γραμμένη σε Lua και η Theano που, όπως και το TensorFlow, είναι κατ' αρχήν μια Python βιβλιοθήκη. Για μια αναλυτική σύγκριση μεταξύ των βιβλιοθηκών που υπάρχουν για deep learning μπορεί κανείς να ανατρέξει εδώ, [16]. Καθοριστικό ρόλο στην επιλογή παίζει το community καθώς και τα projects που έχουν ήδη αναπτυχθεί σε μια βιβλιοθήκη, και εδώ το TensorFlow φαίνεται πως υπερέχει.

Το TensorFlow παρέχει το API του σε Python – και σύντομα σε σταθερές εκδόσεις για C++ , Java, Go [17] – για τη δημιουργία του γράφου ροής δεδομένων (dataflow graph), τον ορισμό δηλαδή των εισόδων και των μαθηματικών λειτουργιών μεταξύ τους. Ο πυρήνας (core) της βιβλιοθήκης είναι γραμμένος κυρίως σε C++, γεγονός που μπορεί να εξασφαλίσει την υπολογιστική αποδοτικότητά του. Στις επόμενες παραγράφους θα δούμε τα βασικά σημεία στον χειρισμό της βιβλιοθήκης για την ανάπτυξη deep learning μοντέλων.

Σημειώνεται ότι στην παρούσα εργασία, έχει χρησιμοποιηθεί Python 3 και TensorFlow 1.2.1 για την ανάπτυξη των μοντέλων.

4.2 Δομικά στοιχεία του TensorFlow

Στην παράγραφο αυτή παρουσιάζονται οι βασικότερες έννοιες του TensorFlow. Αρκετές λεπτομέρειες ή παρατηρήσεις συμπληρώνονται στην υλοποίηση των μοντέλων.

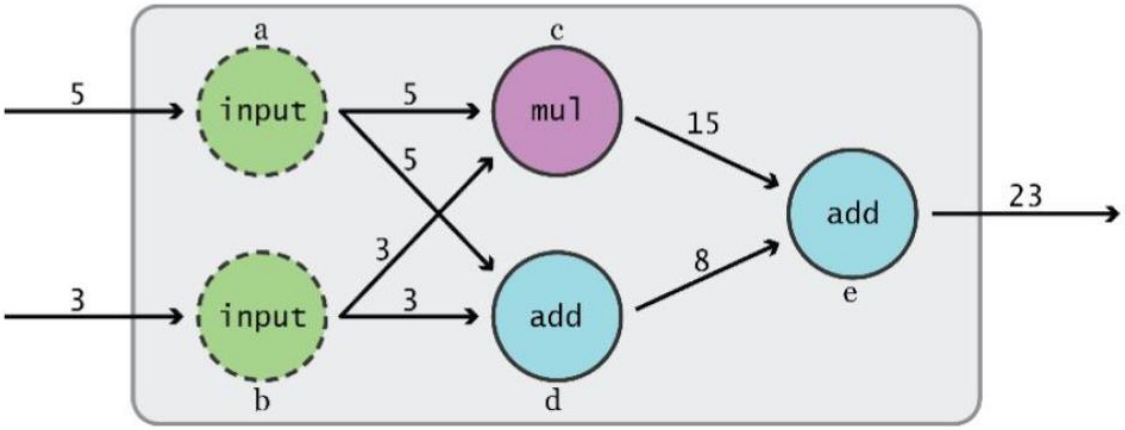
4.2.1 Γράφοι και Sessions

Όπως αναφέρθηκε, με το TensorFlow API ορίζουμε τον γράφο του μοντέλου μας. Οι λειτουργίες του dataflow graph θα εκτελεστούν στη συνέχεια από ένα Session object, στο οποίο εμπεριέχονται. Με αυτόν τον τρόπο υπολογίζονται τελικά τα Tensor objects.

Προτού δούμε λεπτομερέστερα τις παραπάνω έννοιες, επισημαίνουμε ότι η αναπαράσταση των υπολογισμών ως γράφος ροής δεδομένων (dataflow graph) έχει τα ακόλουθα πλεονεκτήματα:

1. Εξοικονόμηση υπολογιστικής ισχύος (εκτελούνται μόνο οι υπο-γράφοι που οδηγούν στις τιμές που επιθυμούμε να βρούμε)
2. Διαίρεση των υπολογισμών σε μικρά, διαφορετικά μέρη
3. Διευκόλυνση της υλοποίησης κατανεμημένων υπολογισμών με τον επιμερισμό τους σε πολλαπλές CPUs, GPUs ή συσκευές
4. Αρκετά μοντέλα μηχανικής μάθησης περιγράφονται συχνά ως κατευθυνόμενοι γράφοι (directed graphs)

Στην παρακάτω εικόνα παρουσιάζεται ένας απλός γράφος ροής δεδομένων:



Εικόνα 4.1: Γράφος ροής δεδομένων

Παρατηρούμε ότι είναι σαφώς ορισμένες οι είσοδοι, βαθμωτά μεγέθη στην απλή περίπτωση εδώ, καθώς και οι μαθηματικές λειτουργίες οι οποίες θα γίνουν. Στη γενική περίπτωση οι είσοδοι είναι tensors, n-διάστατοι πίνακες δηλαδή όπου για $n = 0$ έχουμε ένα βαθμωτό μέγεθος (έναν αριθμό), για $n = 1$ έχουμε ένα διάνυσμα, για $n = 2$ έναν δισδιάστατο πίνακα και ούτω καθεξής.

Περνάμε σε κώδικα Python ώστε να δούμε πως ορίζουμε έναν απλοϊκό γράφο στο TensorFlow, ο οποίος θα λάβει δύο αριθμούς σαν είσοδο και θα τους προσθέσει. Αρχικά, συμπεριλαμβάνουμε τη βιβλιοθήκη και ορίζουμε ακολούθως:

```
import tensorflow as tf
a = tf.add(3, 5)
```

Μέχρι εδώ έχουμε απλά ορίσει το γράφο μας. Για να εκτελεστούν οι λειτουργίες του και να πάρουμε την τιμή του a, θα πρέπει να δημιουργήσουμε ένα Session και μέσα σε αυτό να δώσουμε την εντολή ώστε να υπολογιστεί το a:

```
import tensorflow as tf
a = tf.add(3, 5)
with tf.Session() as sess:
    print(sess.run(a))
```

Το παραπάνω πρόγραμμα μας δίνει την τιμή 8 στην έξοδό του. Αυτό που κάναμε ήταν ότι δημιουργήσαμε ένα Session object, το οποίο περιέχει το περιβάλλον στο οποίο εκτελούνται τα Operation objects (μαθηματικές λειτουργίες) και υπολογίζονται τα Tensor objects. [18]

4.2.2 Βασικές Λειτουργίες και τύποι του TensorFlow

Στο παραπάνω πρόγραμμα, περάσαμε κατευθείαν τις εισόδους μας χωρίς να ορίσουμε τι tensor είναι. Από τη στιγμή που θέλουμε constant tensor, θα γράψουμε το πρόγραμμά μας:

```
import tensorflow as tf
a = tf.constant(3)
b = tf.constant(5)
x = tf.add(a, b)
with tf.Session() as sess:
    print(sess.run(x))
```

Εδώ ορίσαμε ότι οι είσοδοι a, b είναι constant tensors και τους δώσαμε τις τιμές τους. Θα μπορούσαμε ακόμα να ορίσουμε τον τύπο των στοιχείων των a και b καθώς και τις διαστάσεις τους με επιπλέον ορίσματα που είναι διαθέσιμα στην tf.constant().

Είδαμε ακόμα ότι στις εισόδους εφαρμόσαμε τη λειτουργία add() που παρέχει η βιβλιοθήκη. Παρακάτω φαίνονται κάποια παραδείγματα γενικών λειτουργιών με εισόδους διανύσματα:

```
a = tf.constant([3, 6])
b = tf.constant([2, 2])
tf.add(a, b) # >> [5 8]
tf.add_n([a, b, b]) # >> [7 10]. Equivalent to a + b + b
tf.mul(a, b) # >> [6 12] because mul is element wise
tf.matmul(a, b) # >> ValueError
tf.matmul(tf.reshape(a, [1, 2]), tf.reshape(b, [2, 1])) # >> [[18]]
tf.div(a, b) # >> [1 3]
tf.mod(a, b) # >> [1 0]
```

Στα σχόλια η έξοδος κάθε λειτουργίας. Επίσης, έχουμε τη δυνατότητα να ορίσουμε constant value tensors, με μηδενικά για παράδειγμα, tensors που οι τιμές τους είναι κάποια ακολουθία ή τυχαία tensors που οι τιμές τους ακολουθούν κάποια κατανομή, όπως την κανονική. [19]

Πέρα από τα constant tensors, ασφαλώς και χρειαζόμαστε variables στις οποίες θα μπορούμε να περνάμε τιμές και να τις υπολογίζουμε. Ένα παράδειγμα φαίνεται παρακάτω:

```
W = tf.Variable(10)
assign_op = W.assign(100)
with tf.Session() as sess:
```

```
    sess.run(W.initializer)
    sess.run(assign_op)
    print(W.eval()) # >> 100
```

Ορίσαμε μια μεταβλητή με την τιμή 10 και στη συνέχεια την ανάθεση σε αυτήν της τιμής 100. Η δημιουργία της μεταβλητής έγινε εντός του Session μέσω του initializer και η ανάθεση με την εκτέλεση της αντίστοιχης λειτουργίας. Τέλος, ο υπολογισμός της μεταβλητής γίνεται και πάλι εντός Session με την κλήση της eval().

Πολλές φορές χρειαζόμαστε να μπορούμε να ορίσουμε tensors που θα συμπεριληφθούν στον γράφο, χωρίς να ξέρουμε εξαρχής την τιμή τους. Εδώ έρχονται τα placeholders στα οποία θα περάσουμε τιμές εντός του Session μέσω ενός dictionary, όπως φαίνεται στο παράδειγμα παρακάτω:

```
import tensorflow as tf
import numpy as np

x = tf.placeholder(tf.float32, shape=(1024, 1024))
y = tf.matmul(x, x)

with tf.Session() as sess:
    rand_array = np.random.rand(1024, 1024)
    print(sess.run(y, feed_dict={x: rand_array}))
```

Ορίσαμε εδώ ένα placeholder x που περιμένει τιμές τύπου float32 και έχει ορισμένες διαστάσεις, στο οποίο περάσαμε την τιμή που υπολογίσαμε εντός Session μέσω του dictionary feed_dict. Είδαμε επίσης ότι χρησιμοποιήσαμε τη βιβλιοθήκη numpy για να κάνουμε generate τις τιμές που θέλαμε να περάσουμε στο placeholder.

Μια τελευταία λειτουργία που έχει αξία να αναφέρουμε, είναι η δυνατότητα να αποθηκεύουμε τις τιμές των μεταβλητών. Θα αναπτύξουμε μοντέλα που θα εκπαιδεύουμε και οι τιμές των παραμέτρων θα αλλάζουν διαρκώς. Ως επί το πλείστον, θα χρειάζεται αρκετός χρόνος για να φτάσουμε σε ένα καλό σημείο την εκπαίδευσή μας, που σημαίνει ότι θα θέλουμε να τη συνεχίσουμε από εκεί που την αφήσαμε, και ασφαλώς θα θέλουμε να αποθηκεύσουμε εν τέλει το μοντέλο μας. Μπορεί, ακόμα, να επιθυμούμε να κρατήσουμε κάποιο checkpoint στην εκπαίδευσή μας στο οποίο να έχουμε τη δυνατότητα να επανέλθουμε. Όλη αυτή η διαχείριση των πειραμάτων μπορεί να γίνει με την κλάση tf.train.Saver [20].

Ένα παράδειγμα χρησιμοποίησής της φαίνεται παρακάτω:

```
# Create a saver.
saver = tf.train.Saver()
# Launch the graph and train, saving the model every 1,000 steps.
sess = tf.Session()

# Restore model, if there is a previously saved model
if tf.train.latest_checkpoint(model_path) != None:
    saver.restore(sess, tf.train.latest_checkpoint(model_path))
    print("Model restored")

for i in range(epochs):
    sess.run(training_op)
```

```
    if i % 5 == 0:
        saver.save(sess, model_path)
        print("Model saved")
```

Ορίζοντας ένα Saver object, μπορούμε να κάνουμε restore το μοντέλο μας δίνοντας το path στο οποίο έχει πρωτύτερα αποθηκευτεί. Επίσης, έχοντας ορίσει το σύνολο των εποχών, αποθηκεύουμε εδώ τις τιμές των μεταβλητών ανά 5 εποχές. Το παραπάνω δεν είναι πλήρως λειτουργικός κώδικας καθώς μένει να οριστούν τα model_path, training_op και epochs. Ακόμα, μπορούμε να περάσουμε και άλλα ορίσματα στη μέθοδο save() ώστε να αποθηκεύουμε κάθε checkpoint σε άλλο αρχείο και να μην κάνουμε overwrite το ίδιο, όπως στην περίπτωση παραπάνω.

Κλείνοντας, να αναφέρουμε ότι στην ανάπτυξη των μοντέλων που ακολουθεί θα αναφερθούν αρκετές ακόμα πληροφορίες για την χρήση της βιβλιοθήκης. Με τα παραπάνω, θα πρέπει να έχουμε ήδη κατανοήσει τη βασική δομή και τον τρόπο που χρησιμοποιούμε το TensorFlow για τους υπολογισμούς μας.

4.3 Deep feedforward NN για την αναγνώριση ψηφίων γραμμένων με το χέρι

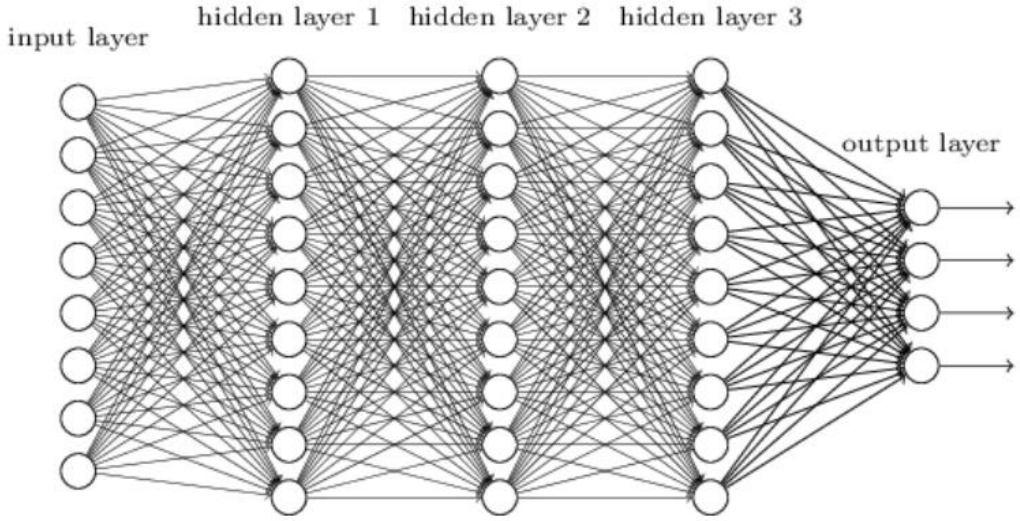
Έχοντας υπόψη μας τις βασικές έννοιες του TensorFlow, θα υλοποιήσουμε ένα deep feedforward νευρωνικό δίκτυο για την αναγνώριση αριθμητικών ψηφίων που έχουν γραφτεί με το χέρι. Τα δεδομένα μας θα τα αντλήσουμε από τη γνωστή MNIST database (Modified National Institute of Standards and Technology database) η οποία παρέχει δεκάδες χιλιάδες ψηφία γραμμένα με το χέρι και χρησιμοποιείται πολύ συχνά για εκπαίδευση μοντέλων στη μηχανική μάθηση. Η βάση δεδομένων εμπεριέχει εικόνες $28\text{x}28\ pixel$ όπου κάθε pixel έχει κάποια τιμή στην κλίμακα του γκρι (grayscale). Παρακάτω, φαίνεται ένα δείγμα από τις εικόνες αυτές:

Εικόνα 4.2: Δείγμα ψηφίων που περιέχονται στη MNIST database[21]

Στόχος είναι, με είσοδο μία εικόνα ενός ψηφίου γραμμένου με το χέρι, το νευρωνικό δίκτυο να αναγνωρίζει το ψηφίο αυτό, να την ταξινομεί δηλαδή στη σωστή κατηγορία στο σύνολο των 10 κατηγοριών (ψηφία 0-9). Για τον σκοπό αυτό, θα φτιάξουμε ένα

feedforward νευρωνικό δίκτυο με βαθιά αρχιτεκτονική. Η θεωρία που διέπει το δίκτυο αυτό έχει αναπτυχθεί στην παράγραφο 2.3 της εργασίας. Συνεπώς, εδώ θα δοθεί έμφαση στην υλοποίηση του δικτύου στο TensorFlow.

Με τον όρο feedforward NN εννοούμε το δίκτυο στο οποίο οι συνδέσεις μεταξύ των νευρώνων δεν σχηματίζουν κάποιο κύκλο, όπως στα RNN. Με την ακριβή χρήση του όρου, τα CNNs είναι επίσης feedforward δίκτυα αφού η πληροφορία μεταβιβάζεται πάντοτε από τα προηγούμενα στα επόμενα στρώματα. Στα πλαίσια της εργασίας, αναφέρουμε τα Convolutional Neural Networks με το όνομά τους και χρησιμοποιούμε τον όρο feedforward network ως συντόμευση του όρου fully-connected feedforward network. Πρόκειται ουσιαστικά για ένα κλασικό πολυστρωματικό νευρωνικό δίκτυο, όπως φαίνεται παρακάτω:

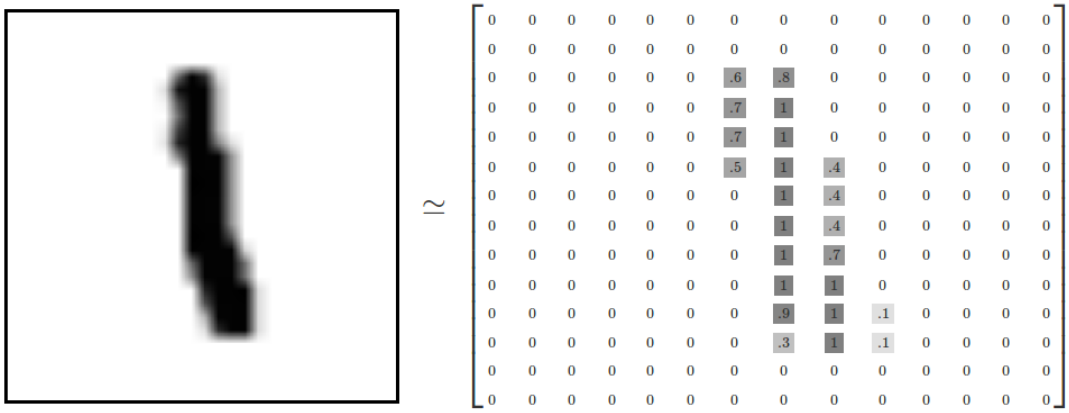


Εικόνα 4.3: Fully-connected deep FF νευρωνικό δίκτυο [11, Κεφ. 5]

Το δίκτυο αυτό αποτελείται από ένα input layer, από κάποια hidden layers και το output layer. Κατά κανόνα, θεωρούμε βαθιά (deep) νευρωνικά δίκτυα εκείνα που έχουν περισσότερα από ένα κρυφά στρώματα.

Ξεκινάμε μελετώντας πλήρως τα δεδομένα μας. Όπως έχει αναφερθεί, θα χρησιμοποιήσουμε το MNIST dataset από εδώ, [22], το οποίο περιέχει 55000 ψηφία στο training set, 10000 στο test set και άλλα 5000 στο validation set. Στην εργασία αυτή, οι τιμές των υπερ-παραμέτρων των νευρωνικών δικτύων έχουν αντληθεί από σχετικές δημοσιεύσεις σε επιστημονικά περιοδικά ή στο διαδίκτυο. Για τον λόγο αυτό, δεν έχει χρησιμοποιηθεί το validation set που θα υπηρετούσε τον σκοπό αυτό. Ο λόγος ύπαρξης των τριών αυτών διαφορετικών datasets έχει αναφερθεί λεπτομερώς στην παράγραφο 3.1 της παρούσας εργασίας.

Συνεπώς, θα έχουμε στη διάθεσή μας 55000 εικόνες για την εκπαίδευση του δικτύου διάστασης $28\text{x}28\ pixel$ η κάθε μία. Κάθε pixel έχει μία τιμή που κυμαίνεται από 0 έως 1 και καθορίζει το επίπεδό του στην κλίμακα του γκρι. Οι απεικονίσεις στην εργασία αυτή έχουν γίνει θεωρώντας ως 0 το λευκό χρώμα και ως 1 το μαύρο. Η αντίθετη θεώρηση μπορεί κάλλιστα να γίνει, απολύτως συμμετρικά. Ένα παράδειγμα των εικόνων αυτών φαίνεται παρακάτω:



Εικόνα 4.4: Ο αριθμός 1 γραμμένος με το χέρι και ο αντίστοιχος πίνακας 28x28 [23]

Στην εικόνα βλέπουμε πως το γραμμένο με το χέρι ψηφίο μεταφράζεται στον πίνακα 28x28 με τιμές από 0 έως 1 σε κάθε στοιχείο του. Για να δώσουμε κάθε τέτοια εικόνα ως είσοδο στο νευρωνικό μας δημιουργούμε ένα διάνυσμα από $28\text{x}28 = 784$ αριθμούς. Δεν έχει σημασία ο τρόπος με τον οποίο θα μετασχηματίσουμε τον πίνακα σε διάνυσμα, αρκεί να κάνουμε το ίδιο για όλες τις εικόνες.

Αφού έχουμε ξεκαθαρίσει με τα data μας, γνωρίζουμε ήδη τη διάσταση του input layer του feedforward νευρωνικού μας δικτύου. Θα αποτελείται από 784 νευρώνες, ο καθένας από τους οποίους δεν κάνει κάτι άλλο από το να μεταφέρει την τιμή που έχει το διάνυσμα στο σημείο εκείνο σε όλους τους νευρώνες του $1ου$ κρυφού στρώματος, πολλαπλασιασμένη με το αντίστοιχο βάρος.

Ακόμα, κάθε εικόνα φέρει μαζί της και μια ετικέτα (label), που δηλώνει την κατηγορία στην οποία ανήκει, ποιο ψηφίο είναι δηλαδή. Πρόκειται για ένα πρόβλημα ταξινόμησης με 10 κατηγορίες. Συνεπώς, θα χρησιμοποιήσουμε τη συνάρτηση softmax, βλέπε παράγραφο 2.5, στο output layer με 10 νευρώνες καθένας από τους οποίους θα δηλώνει τη πιθανότητα το ψηφίο να ανήκει στην αντίστοιχη κατηγορία από το 0 έως 9. Το ψηφίο θα ταξινομείται στην κατηγορία με τη μεγαλύτερη πιθανότητα.

Γνωρίζουμε, λοιπόν, ότι το στρώμα εισόδου θα έχει 784 νευρώνες και το στρώμα εξόδου 10. Η συνάρτηση ενεργοποίησης στο στρώμα εξόδου θα είναι η softmax. Μένει να επιλέξουμε το πλήθος των κρυφών στρωμάτων, τον αριθμό των νευρώνων που θα έχει το καθένα, τη συνάρτηση ενεργοποίησής τους καθώς και τη συνάρτηση κόστους για τον αλγόριθμο εκμάθησης μαζί με κάποιες ακόμα υπερ-παραμέτρους.

Από ένα πολύ σημαντικό paper, που ήταν ένα από αυτά που εισήγαγαν το dropout για regularization, [24], επιλέγουμε να υλοποιήσουμε αρχικά μια αρχιτεκτονική με 2 hidden layers αποτελούμενα από 2000 νευρώνες το καθένα με συνάρτηση ενεργοποίησης το rectifier. Οπότε, θα έχουμε 784 νευρώνες στο στρώμα εισόδου, 2 κρυφά στρώματα με 2000 ReLUs το καθένα και το στρώμα εξόδου με 10 νευρώνες.

Σχετικά με τις υπόλοιπες υπερ-παραμέτρους, επιλέγουμε αλγόριθμο gradient descent με cross-entropy συνάρτηση κόστους, μέγεθος κάθε mini-batch ίσο με 100 παραδείγματα, learning rate 0.01 και εκπαιδεύουμε το μοντέλο μας για 600 εποχές.

Ας δούμε πως θα υλοποιήσουμε το νευρωνικό δίκτυο στο TensorFlow. Αρχικά, θα αντλήσουμε τα δεδομένα μέσω της βιβλιοθήκης:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data
```

```
# input data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
trX, trY, teX, teY = mnist.train.images, mnist.train.labels, mnist.test.images,
mnist.test.labels
```

Εκχωρούμε στο αντικείμενο mnist το dataset και στη συνέχεια δημιουργούμε τα arrays που περιέχουν τα διανύσματα εκπαίδευσης μαζί με τις ετικέτες τους καθώς και τα αντίστοιχα του συνόλου ελέγχου.

Ορίζουμε, επίσης, συναρτήσεις που θα χρησιμοποιήσουμε στην ανάπτυξη του μοντέλου και δομούν με καλύτερο τρόπο τον κώδικά μας:

```
def init_weights(shape):
    return tf.Variable(tf.random_normal(shape, stddev=0.01))

def init_biases(shape):  # slightly positive initial bias to avoid "dead" neurons
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

def hidden_layer(X, w_h1, b_h1):
    return (tf.nn.relu(tf.matmul(X, w_h1) + b_h1))

def out_layer(h, w_o, b_o):
    return (tf.matmul(h, w_o) + b_o) # note that we don’t take the softmax at the
end because our cost includes it
```

Παρατηρούμε ότι χρησιμοποιούμε παντού μεθόδους από τη βιβλιοθήκη TensorFlow καθώς επίσης για από το module tf.nn για την ανάπτυξη νευρωνικών δικτύων [25]. Έπειτα, δίνουμε τιμή στις υπερ-παραμέτρους του δικτύου μας:

```
# define hyper-parameters' values
number_of_hidden_layers = 2
neurons_in_hidden_layer = 2000
mini_batch_size = 100
learning_rate = 0.01
epochs = 600
```

Και διαμορφώνουμε το μοντέλο μας, ορίζουμε δηλαδή τον γράφο ροής των δεδομένων:

```
X = tf.placeholder("float", [None, 784])
Y = tf.placeholder("float", [None, 10])

w_h = []
b_h = []

for j in range(number_of_hidden_layers):
    if j == 0:
        w_h.append(init_weights([784, neurons_in_hidden_layer]))
```

```
        b_h.append(init_biases([neurons_in_hidden_layer]))
    else:
        w_h.append(init_weights([neurons_in_hidden_layer,
neurons_in_hidden_layer]))
        b_h.append(init_biases([neurons_in_hidden_layer]))

w_o = init_weights([neurons_in_hidden_layer, 10])
b_o = init_biases([10])

h = []

for l in range(number_of_hidden_layers):
    if l == 0:
        h.append(hidden_layer(X, w_h[l], b_h[l]))
    else:
        h.append(hidden_layer(h[-1], w_h[l], b_h[l]))

y_bef_softmax = out_layer(h[-1], w_o, b_o)


# define cost function
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_bef_softmax,
labels=Y))

# construct an optimizer
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# classify input image
predict_op = tf.argmax(y_bef_softmax, 1)
```

Στη συνέχεια, δημιουργούμε ένα session object, αρχικοποιούμε όλες τις μεταβλητές, και υπολογίζουμε τα Tensors που μας ενδιαφέρουν από τον γράφο:

```
# Launch the graph in a session
sess = tf.Session()

# initialize all variables
sess.run(tf.global_variables_initializer())

for i in range(epochs):
    for start, end in zip(range(0, len(trX), mini_batch_size),
range(mini_batch_size, len(trX)+1, mini_batch_size)):
        sess.run(train_op, feed_dict={X: trX[start:end], Y: trY[start:end]})

    wrong_classified = np.sum(np.argmax(teY, axis=1) != sess.run(predict_op,
feed_dict={X: teX}))

    print(i, wrong_classified)
```

Για να το δούμε κάπως αναλυτικά, ορίζουμε τις μεταβλητές εισόδου και εξόδου με τις αντίστοιχες διαστάσεις και τους τύπους τους, δημιουργούμε δύο δομές που θα περιέχουν τα βάρη και τα biases των κρυφών στρωμάτων και αρχικοποιούμε στις κατάλληλες διαστάσεις τους. Έπειτα στη δομή h, έχουμε όλες τις εξόδους των κρυφών στρωμάτων και παίρνουμε τελικά την έξοδο του στρώματος εξόδου πριν τη συνάρτηση softmax, η οποία συμπεριλαμβάνεται στην cross-entropy συνάρτηση κόστους. Χρησιμοποιούμε gradient descent για την εκπαίδευση και ταξινομούμε το πρότυπο εισόδου στην κατηγορία που έχει την μεγαλύτερη τιμή στην έξοδο πριν τη softmax. Η softmax είναι αύξουσα οπότε δεν επηρεάζεται κάπου η επιλογή της μέγιστης τιμής, πριν ή μετά από αυτή.

Αφού έχουμε καθορίσει πλήρως το dataflow graph, τροφοδοτούμε το μοντέλο μας με ένα νέο batch προτύπων κάθε φορά μέχρι να παρουσιαστούν και τα 55000 σε κάθε εποχή. Πριν συνεχίσουμε στην επόμενη εποχή, ελέγχουμε την παρούσα απόδοση του μοντέλου μας μέσω του test set και τυπώνουμε στην έξοδο το πλήθος των προτύπων που ταξινομήθηκαν σε λάθος κατηγορία.

Το παραπάνω μοντέλο είναι πλήρως λειτουργικό και ικανό να εκπαιδευτεί. Αυτό που σίγουρα επιθυμούμε είναι να αποθηκεύουμε την έξοδό του σε κάποιο (txt) αρχείο, ώστε να έχουμε τη δυνατότητα να παρουσιάσουμε γραφικά τα αποτελέσματα ή να τα συγκρίνουμε με κάποιο άλλο μοντέλο. Ακόμα, η παραπάνω εκτέλεση χρειάζεται, ανάλογα τους υπολογιστικούς πόρους, κάποιες ώρες να ολοκληρωθεί. Για τον λόγο αυτό, χρειαζόμαστε να αποθηκεύουμε το μοντέλο μας σε τακτά χρονικά διαστήματα για να έχουμε τη δυνατότητα να διακόψουμε την εκπαίδευση και να συνεχίσουμε μετά από το ίδιο σημείο, όπως έχει ήδη αναφερθεί στο τέλος της παραγράφου 4.2.2. Οι υλοποιήσεις αυτές αφήνονται στον ενδιαφερόμενο να τις αναπτύξει.

4.4 Ενσωμάτωση Dropout σε feedforward NN και πειραματικά αποτελέσματα

Για να γίνει εμφανής η επίδραση του dropout στην απόδοση του δικτύου, εκτελέσαμε το εξής πείραμα. Κρατήσαμε όλες τις άλλες παραμέτρους σταθερές και εκπαιδεύσαμε το δίκτυο αρχικά χωρίς regularization, στη συνέχεια με L^2 εξομάλυνση, έπειτα με dropout 50% στους νευρώνες των κρυφών στρωμάτων και τέλος με dropout 50% στα κρυφά στρώματα και 20% στο στρώμα εισόδου.

Η υλοποίηση χωρίς εξομάλυνση υπάρχει στην προηγούμενη παράγραφο. Στην περίπτωση που θα ενσωματώσουμε L^2 regularization, θα πρέπει να μεταβάλλουμε την συνάρτηση κόστους ώστε να περιέχει τον όρο εξομάλυνσης. Ο συντελεστής του όρου αυτού, lamda, επιλέχθηκε εδώ ίσος με 0.001. Η υλοποίηση της συνάρτησης κόστους φαίνεται παρακάτω:

```
# define cost function with L2 Regularization
cost = tf.nn.softmax_cross_entropy_with_logits(logits=y_bef_softmax, labels=Y)

for k in range(number_of_hidden_layers):
    if k == 0:
        regularizers = tf.nn.l2_loss(w_h[k])
    else:
        regularizers = regularizers + tf.nn.l2_loss(w_h[k])

cost = tf.reduce_mean(cost + lamda * regularizers)
```

Στην περίπτωση που θα χρησιμοποιήσουμε dropout, επαναφέρουμε αρχικά τη συνάρτηση κόστους στη προηγούμενη μορφή της:

```
# define cost function
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_bef_softmax,
labels=Y))
```

Ορίζουμε τις ακόλουθες υπερ-παραμέτρους και τα placeholders:

```
# dropout hyper-parameters, set 0.0 for no dropout
dropout_input = 0.2
dropout_hidden = 0.5

# define placeholders for Dropout
keep_prob_hidden = tf.placeholder(tf.float32)
keep_prob_input = tf.placeholder(tf.float32)
```

Ενσωματώνουμε το dropout στη μέθοδο για τα κρυφά στρώματα:

```
def hidden_layer(X, w_h1, b_h1, keep_prob_hidden):
    h = tf.nn.relu(tf.matmul(X, w_h1) + b_h1)
    return (tf.nn.dropout(h, keep_prob_hidden))
```

Τροποποιούμε το μοντέλο μας ώστε να δέχεται dropout και στο στρώμα εισόδου, όπως φαίνεται παρακάτω:

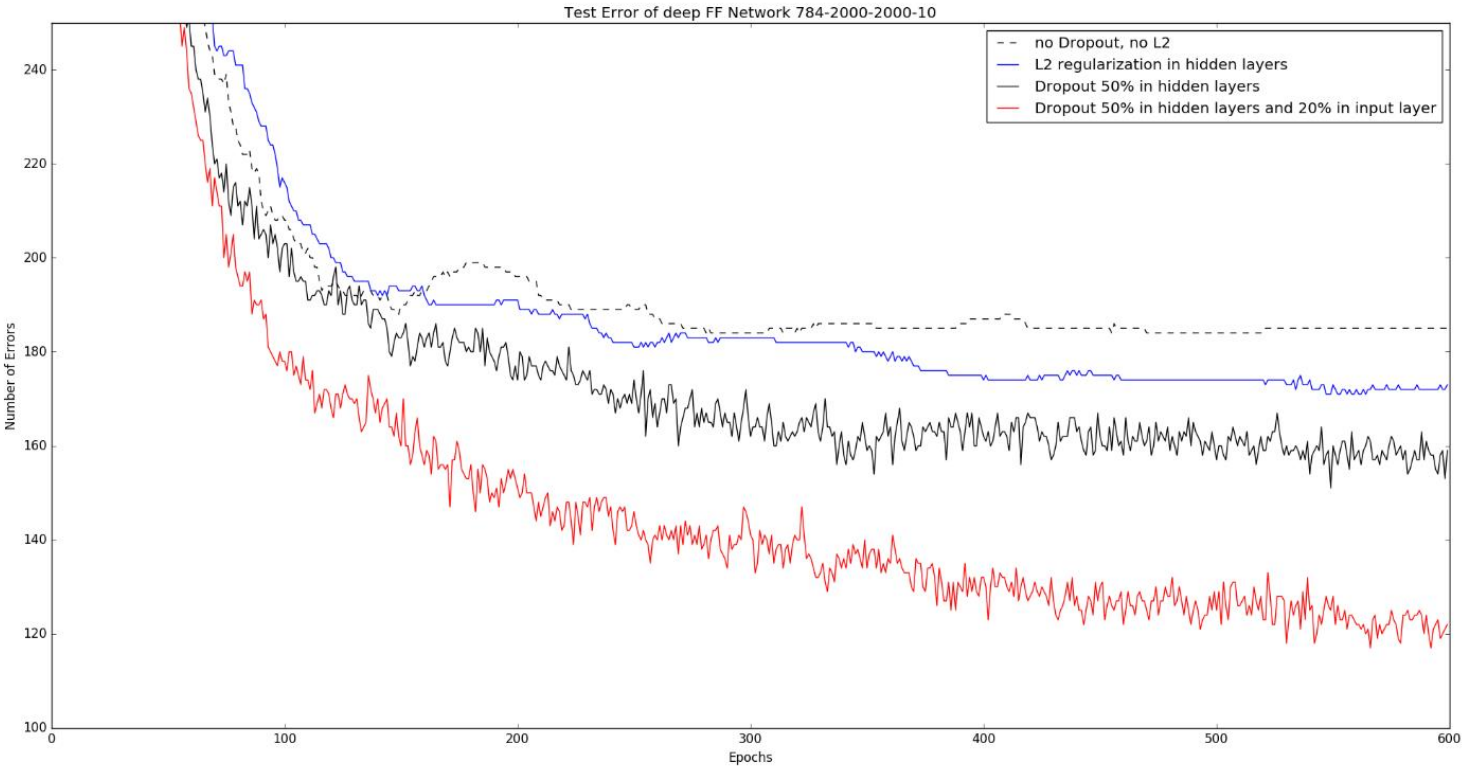
```
# Dropout for regularization
X_drop = tf.nn.dropout(X, keep_prob_input)

h = []

for l in range(number_of_hidden_layers):
    if l == 0:
        h.append(hidden_layer(X_drop, w_h[l], b_h[l], keep_prob_hidden))
    else:
        h.append(hidden_layer(h[-1], w_h[l], b_h[l], keep_prob_hidden))

y_bef_softmax = out_layer(h[-1], w_o, b_o)
```

Τέλος, παρέχουμε τις υπερ-παραμέτρους του dropout μέσω του dictionary στην εκτέλεση του session:

```
for i in range(epochs):
    for start, end in zip(range(0, len(trX), mini_batch_size),
range(mini_batch_size, len(trX)+1, mini_batch_size)):
        sess.run(train_op, feed_dict={X: trX[start:end], Y: trY[start:end],
keep_prob_hidden:(1.0 - dropout_hidden), keep_prob_input: (1.0 - dropout_input)})

    wrong_classified = np.sum(np.argmax(teY, axis=1) != sess.run(predict_op,
feed_dict={X: teX, keep_prob_hidden: 1.0, keep_prob_input: 1.0}))

    print(i, wrong_classified)
```

Παρατηρούμε ότι κατά τον έλεγχο της απόδοσης του μοντέλου στο test set, η πιθανότητα διατήρησης κάθε νευρώνα ορίζεται ίση με 1 ώστε να συμμετέχει στο πλήρες νευρωνικό δίκτυο, όπως έχει αναφερθεί στην παράγραφο 3.5. Η κανονικοποίηση των βαρών κατά τη διάρκεια του ελέγχου γίνεται από το TensorFlow μέσω της μεθόδου tf.nn.dropout [26].

Εκπαιδεύουμε το νευρωνικό δίκτυο στις τέσσερις περιπτώσεις που αναφέραμε για 600 εποχές. Σε κάθε εποχή ελέγχουμε την απόδοσή του μετρώντας το πλήθος των λαθών που κάνει στο να κατηγοριοποιήσει σωστά τα 10000 ψηφία που υπάρχουν στο test set. Παίρνουμε την ακόλουθη γραφική παράσταση:



Σχήμα 4.1: Απόδοση του feedforward νευρωνικού δικτύου χωρίς regularization, με L^2, με Dropout στα hidden layers και με Dropout στα hidden layers και στο input layer

Στο παραπάνω γράφημα βλέπουμε τις 600 εποχές στον οριζόντιο άξονα και τα εσφαλμένα ταξινομημένα ψηφία στον κάθετο. Παρατηρούμε ότι τα σφάλματα μειώνονται, εν γένει, σε όλες τις περιπτώσεις προϊόντος του χρόνου. Με διακεκομμένη

γραμμή φαίνεται η απόδοση του μοντέλου χωρίς καμία εξομάλυνση. Η καλύτερη απόδοση που είχε ήταν 184 λάθη από τα 10000 ή ακρίβεια ίση με 98.16%. Εφαρμόζοντας L^2 regularization, βελτιώνουμε την απόδοση του δικτύου ανεβάζοντας την απόδοσή του στο 98.29%. Πιθανόν, θα μπορούσαμε να μειώσουμε ακόμα περισσότερο τα σφάλματα με την εύρεση κάποιας αποδοτικότερης τιμής για τον συντελεστή λ του όρου εξομάλυνσης.

Στη συνέχεια, εφαρμόζουμε dropout. Αρχικά, στα κρυφά στρώματα με πιθανότητα επιβίωσης 0.5 για κάθε νευρώνα. Παρατηρούμε αισθητή βελτίωση στην γενική τάση της καμπύλης με την απόδοση του νευρωνικού να φτάνει το 98.49%. Βλέπουμε, ακόμα, ότι η διακύμανση της καμπύλης αυτής είναι μεγαλύτερη σε σχέση με τις προηγούμενες. Αυτό είναι κάτι που μπορούμε να ερμηνεύσουμε. Με τη χρήση του τυχαίου dropout, πολλά διαφορετικά απομειωμένα δίκτυα εκπαιδεύονται σε κάθε εποχή. Αυτό έχει σαν αποτέλεσμα η εκπαίδευση να μην μπορεί να λάβει μια συγκεκριμένη τάση σε σύνολο λίγων εποχών. Ο αλγόριθμος ενημερώνει τις τιμές των παραμέτρων βλέποντας κάθε φορά ένα διαφορετικό δίκτυο. Τα απομειωμένα δίκτυα που επιλέχθηκαν τυχαία μέσα σε μια εποχή και οδήγησαν στις νέες τιμές των παραμέτρων, δεν είναι καθόλου βέβαιο ότι θα οδηγήσουν σε καλύτερη απόδοση του συνολικού νευρωνικού δικτύου, σε σχέση με την ακριβώς προηγούμενη εποχή. Σε βάθος χρόνου, όμως, η τυχαιότητα αυτή λειτουργεί προς όφελος της εκμάθησης του μοντέλου. Αναμένουμε, πάντως, η ίδια αρχιτεκτονική να χρειάζεται μεγαλύτερο χρόνο εκπαίδευσης, δηλαδή περισσότερες εποχές, όταν εφαρμόζουμε dropout. [13, σ 1952]

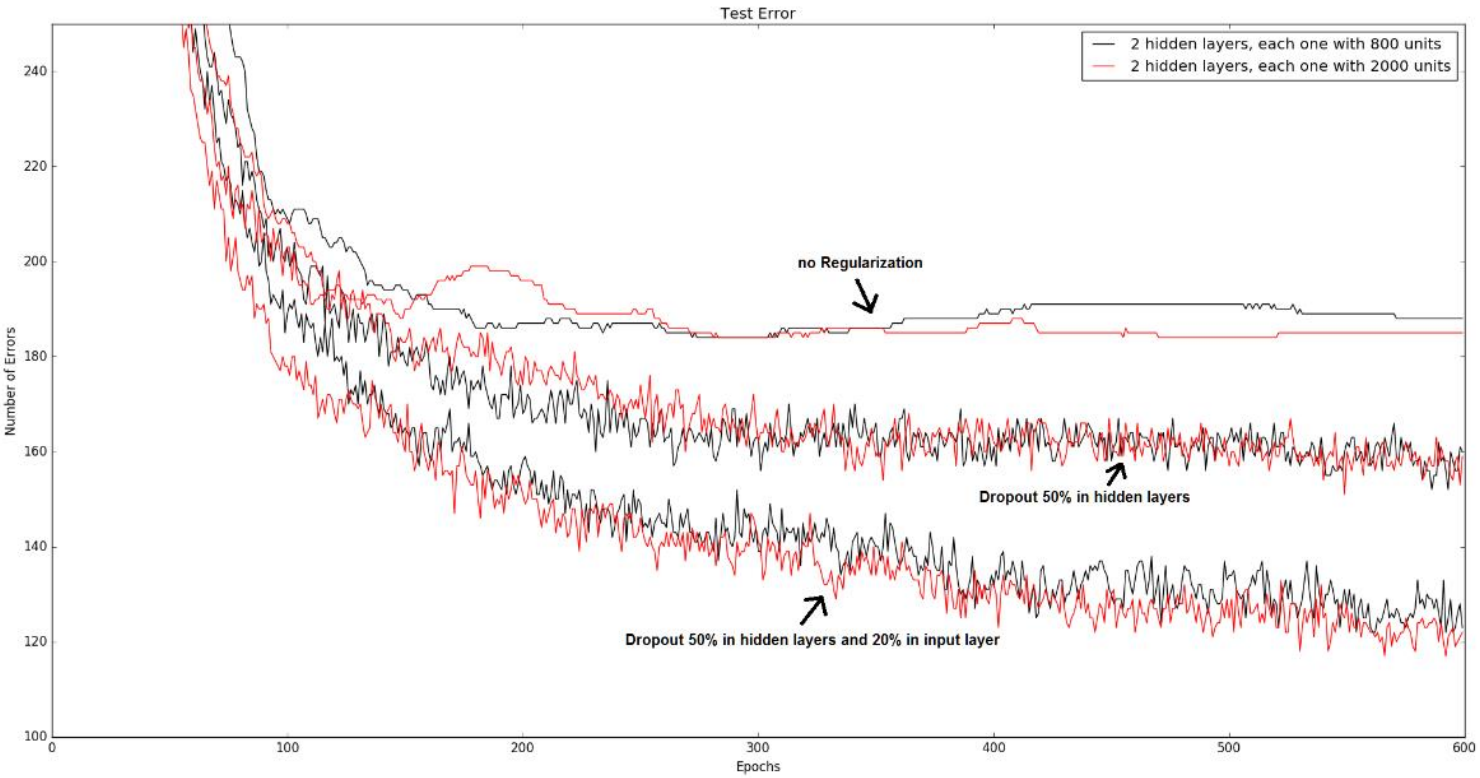
Στην καμπύλη με το κόκκινο χρώμα, βλέπουμε την απόδοση της ίδιας αρχιτεκτονικής όταν πέρα από τα κρυφά στρώματα, εφαρμόσουμε dropout στους νευρώνες του στρώματος εισόδου με πιθανότητα επιβίωσης 0.8 ή όπως λέμε dropout 20%. Εδώ η απόδοση βελτιώνεται πολύ μειώνοντας τα λάθη σε 117 και επιτυγχάνοντας ακρίβεια 98.83% στο test set.

Από τα παραπάνω προκύπτει σίγουρα το συμπέρασμα ότι το dropout συνέβαλε στην καλύτερη εκπαίδευση του νευρωνικού δικτύου μας. Πού είμαστε, όμως, σε σχέση με τις αποδόσεις που υπάρχουν στη βιβλιογραφία; Συγκρίνοντας με την αντίστοιχη αρχιτεκτονική από εδώ, [24], βλέπουμε ότι τα αποτελέσματα μας έχουν αρκετή συνάφεια. Σε 3000 εποχές η απόδοση του δικτύου με dropout στα hidden layers ήταν περίπου 135 λάθη, σε σύγκριση με 151 στα πειράματά μας, ενώ με εφαρμογή dropout και στο input layer, τα λάθη έπεσαν περίπου στα 105 σε σχέση με τα 117 που είχε το δικό μας μοντέλο. Οι διαφορές αυτές είναι φυσιολογικές καθώς υπάρχει διαφορά στον αριθμό των εποχών εκπαίδευσης, όπως επίσης στη χρήση L^2 περιορισμών στα εισερχόμενα βάρη των νευρώνων των κρυφών στρωμάτων, [24, σ 2], κάτι που εμείς δεν χρησιμοποιήσαμε.

4.5 Σύγκριση της απόδοσης διαφόρων feedforward NN με και χωρίς Dropout

Θέλοντας να μελετήσουμε την επίδραση του dropout στην εκπαίδευση και έχοντας ως αναφορά τη βιβλιογραφία, [24], υλοποιήσαμε διαφορετικές αρχιτεκτονικές feedforward νευρωνικών δικτύων. Κρατήσαμε όλες τις υπόλοιπες υπερ-παραμέτρους ίδιες και εξετάσαμε την απόδοσή τους χωρίς regularization, με dropout 50% στα κρυφά στρώματα καθώς και επιπρόσθετο dropout 20% στο στρώμα εισόδου.

Υλοποιούμε, αρχικά, ένα νευρωνικό δίκτυο και πάλι με 2 κρυφά στρώματα, αλλά με 800 νευρώνες τώρα στο καθένα. Παρατηρούμε στο παρακάτω σχήμα την απόδοσή του σε σχέση με το δίκτυο με τους 2000 νευρώνες σε κάθε κρυφό στρώμα:



Σχήμα 4.2: Σύγκριση απόδοσης δύο feedforward νευρωνικών δικτύων με και χωρίς dropout

Η καλύτερη απόδοση του δικτύου αυτού (784-800-800-10) ήταν 98.78% όταν εφαρμόσαμε dropout στα κρυφά στρώματα και στο στρώμα εισόδου.

Τέλος, πέρα από τις δύο αρχιτεκτονικές που έχουμε ήδη δει, αναπτύσσουμε άλλες δύο. Μία με 2 κρυφά στρώματα και 1200 νευρώνες στο καθένα (784-1200-1200-10), και μία με 3 κρυφά στρώματα με 1200 νευρώνες το καθένα (784-1200-1200-1200-10). Τα παρουσιάζουμε όλα στο ίδιο γράφημα έχοντας παράλληλα ως σημείο αναφοράς την καλύτερη απόδοση που πετύχαμε - με οποιοδήποτε δίκτυο - χωρίς regularization, δηλαδή τα 184 λάθη, σχήμα 4.3.

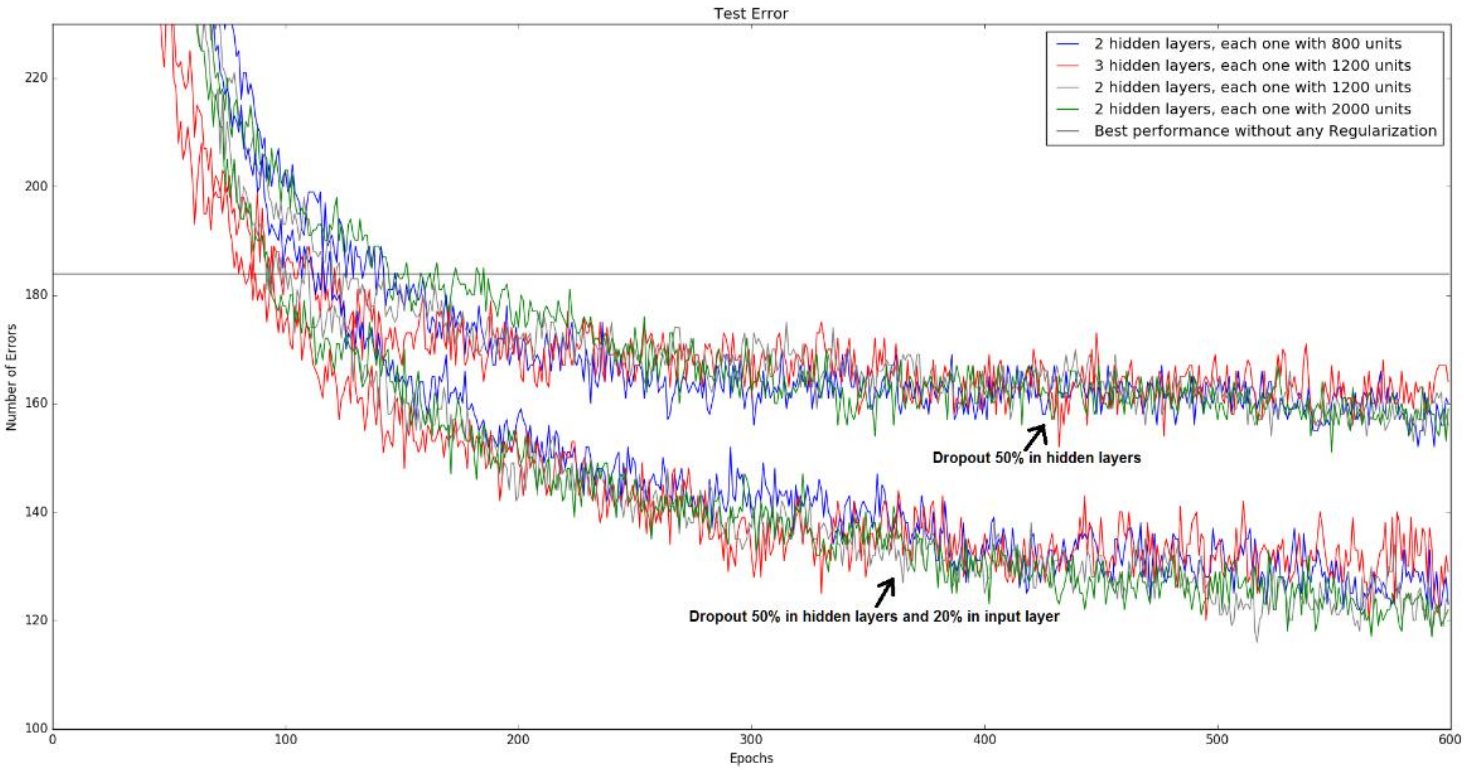
Παρατηρούμε ότι το dropout συμβάλει στη βελτίωση της απόδοσης κάθε νευρωνικού δικτύου με αντίστοιχο τρόπο. Η εφαρμογή του στα κρυφά στρώματα βελτίωσε αισθητά την απόδοση του μοντέλου ενώ ακόμα πιο αισθητή ήταν η αύξηση της απόδοσης με την επιπρόσθετη εφαρμογή του στους νευρώνες του στρώματος εισόδου.

Στον πίνακα που ακολουθεί, παρουσιάζεται αναλυτικά το πλήθος των λανθασμένα ταξινομημένων ψηφίων που είχε κάθε αρχιτεκτονική κατά την εφαρμογή του dropout. Η καλύτερη απόδοση ήταν του δικτύου με 2 κρυφά στρώματα και 1200 νευρώνες το καθένα με ακρίβεια 98.84%.

Ακολούθως, παρουσιάζεται το γράφημα που αναφέρθηκε παραπάνω.

Πίνακας 4.1: Πλήθος λαθών διαφορετικών feedforward νευρωνικών δικτύων με χρήση dropout

Feedforward NN	50% Dropout in hidden layers	50% Dropout in hidden layers + 20% in input layer
2 hidden layers, each one with 800 units	152	122
3 hidden layers, each one with 1200 units	152	120
2 hidden layers, each one with 1200 units	153	**116**
2 hidden layers, each one with 2000 units	151	117



Σχήμα 4.3: Σύγκριση απόδοσης διαφορετικών feedforward NN με εφαρμογή dropout

5. CONVOLUTIONAL NEURAL NETWORKS ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥΣ ΜΕ DROPOUT

5.1 Εισαγωγή στα Convolutional Neural Networks

Τα Convolutional Neural Networks, γνωστά και ως CNNs (Συνελικτικά Νευρωνικά Δίκτυα), είναι ένα είδος νευρωνικών δικτύων ιδιαίτερα κατάλληλων στην επεξεργασία δεδομένων που παρουσιάζουν μια πλεγματοειδή τοπολογία. Το πιο γνωστό παράδειγμα είναι οι εικόνες που μπορούν να λογιστούν σαν ένα δισδιάστατο πλέγμα από pixels. Τα CNNs είναι εξαιρετικά αποδοτικά σε πρακτικές εφαρμογές που έχουν να κάνουν με την επεξεργασία εικόνων και την όραση υπολογιστών. Το όνομα Convolutional υποδηλώνει ότι πρόκειται για νευρωνικά δίκτυα τα οποία χρησιμοποιούν την πράξη της συνέλιξης στη θέση του πολλαπλασιασμού πινάκων, σε τουλάχιστον ένα στρώμα.

Η συνέλιξη (convolution) είναι γενικά μια γραμμική πράξη ανάμεσα σε δύο συναρτήσεις. Για να γίνει κατανοητή η σημασία της, ας αναφέρουμε ένα παράδειγμα. Υποθέτουμε ότι παρακολουθούμε την τροχιά ενός διαστημοπλοίου με έναν αισθητήρα laser. Ο αισθητήρας μας δίνει μια μοναδική έξοδο $x(t)$, τη θέση του διαστημοπλοίου τη χρονική στιγμή t. Έστω, ακόμα, ότι ο αισθητήρας εμπεριέχει θόρυβο και θέλουμε με κάποιο τρόπο να βελτιώσουμε την ακρίβεια των μετρήσεων μας. Αυτό που μπορούμε να κάνουμε είναι να πάρουμε τη μέση τιμή, για κάθε χρονική στιγμή, από ένα σύνολο μετρήσεων που έχουμε λάβει στο πρόσφατο παρελθόν, έχοντας παράλληλα γνώση για την προκαθορισμένη πορεία του διαστημοπλοίου. Βέβαια, οι πιο πρόσφατες μετρήσεις θα είναι πιο σχετικές, οπότε θέλουμε έναν σταθμισμένο μέσο όρο που να δίνει μεγαλύτερη έμφαση στις πιο πρόσφατες παρατηρήσεις. Το επιτυγχάνουμε αυτό με μια συνάρτηση βαρών $w(a)$, όπου a η ηλικία της μέτρησης. Αν εφαρμόσουμε μια τέτοια διαδικασία σταθμισμένου μέσου όρου (weighted averaging) για κάθε χρονική στιγμή, παίρνουμε μια νέα συνάρτηση s που μας παρέχει μια σταθμισμένη εκτίμηση της θέσης του διαστημοπλοίου:

$$s(t) = \int x(a)w(t-a)da$$

Η πράξη αυτή ονομάζεται συνέλιξη και συμβολίζεται συνήθως ως:

$$s(t) = (x * w)(t)$$

Στο παράδειγμά μας, η w θα πρέπει να είναι 0 για όλα τα αρνητικά ορίσματα, αλλιώς θα κοιτάει προς το μέλλον, κάτι που υπερβαίνει τις δυνατότητές μας. Γενικώς, η συνέλιξη ορίζεται για κάθε ζεύγος συναρτήσεων για τις οποίες ορίζεται το παραπάνω ολοκλήρωμα και χρησιμοποιείται και για άλλους λόγους πέραν της εύρεσης του σταθμισμένου μέσου όρου.

Στα CNNs, το αντίστοιχο της συνάρτησης x του παραδείγματός μας, είναι κατά κανόνα η είσοδός μας ενώ αναφέρουμε το αντίστοιχο w ως πυρήνα (kernel) ή φίλτρο. Ακόμα, η έξοδος αναφέρεται συχνά ως ο χάρτης των χαρακτηριστικών (feature map).

Στο παραπάνω παράδειγμά υποθέσαμε έναν αισθητήρα που μπορεί να λαμβάνει μετρήσεις κάθε χρονική στιγμή, κάτι που δεν είναι εφικτό. Συνήθως, όταν εργαζόμαστε με δεδομένα στον υπολογιστή, ο χρόνος θα είναι διακριτός και ο αισθητήρας θα παρέχει δεδομένα κατά τακτά χρονικά διαστήματα, όπως μια μέτρηση το δευτερόλεπτο. Τότε, το χρονικό βήμα t λαμβάνει ακέραιες τιμές και ορίζουμε τη συνέλιξη διακριτού χρόνου:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

Στις εφαρμογές της μηχανικής μάθησης, η είσοδος είναι συνήθως ένας πολυδιάστατος πίνακας δεδομένων και ο πυρήνας ένας πολυδιάστατος πίνακας παραμέτρων που προσαρμόζεται μέσω του αλγορίθμου εκπαίδευσης. Ονομάζουμε τους πίνακες αυτούς tensors. Υποθέτουμε ότι οι συναρτήσεις της εισόδου και του πυρήνα είναι παντού 0 εκτός από τα σημεία εκείνα για τα οποία έχουμε αποθηκευμένες τιμές. Μπορούμε, έτσι, να υλοποιήσουμε το άπειρο άθροισμα σαν το άθροισμα ενός πεπερασμένου αριθμού στοιχείων και να υπολογίσουμε τη συνέλιξη, συχνά σε περισσότερους από έναν άξονες. Θεωρώντας, για παράδειγμα, μια εικόνα ως έναν δισδιάστατο πίνακα εισόδου I και χρησιμοποιώντας έναν επίσης δισδιάστατο πυρήνα K, η συνέλιξη θα είναι:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

Η συνέλιξη ικανοποιεί την αντιμεταθετική ιδιότητα, δηλαδή:

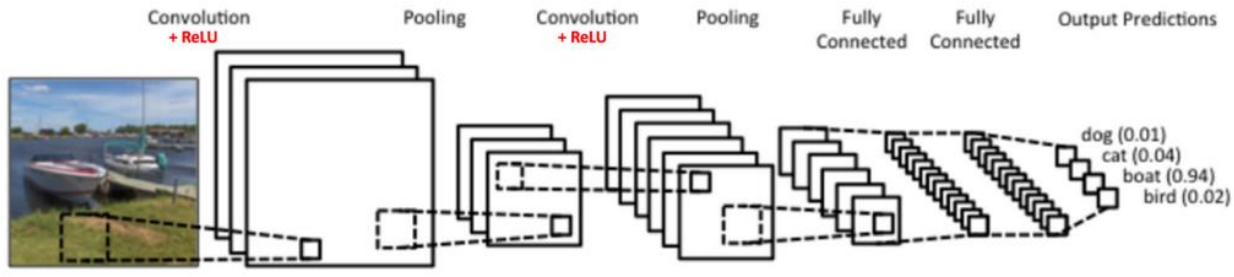
$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n)$$

Πολλές βιβλιοθήκες υλοποιούν με αυτόν τον τρόπο τη συνέλιξη καθώς, λόγω της μικρότερης διασποράς των έγκυρων τιμών για τα m και n, είναι υπολογιστικά πιο αποδοτικός. [7, σσ 330–332]

5.2 Δομή και λειτουργία των Convolutional Neural Networks

Η αρχιτεκτονική LeNet5, [27], ήταν μία από τις πρώτες που κατέστησαν τα CNNs ικανά να συνεισφέρουν πολλά στο πεδίο του deep learning. Η καινοτόμος αυτή εργασία από τον Yann LeCun, ονομάστηκε έτσι μετά από αρκετές επιτυχημένες απόπειρες από το 1988 και μετά. Παρακάτω, παρουσιάζεται με λεπτομέρεια η αρχιτεκτονική αυτή, που εν πολλοίς θα αναπτύξουμε στη συνέχεια στο TensorFlow για να βελτιώσουμε την απόδοσή μας στα δεδομένα από τη MNIST database.

Υπάρχουν αρκετές νέες αρχιτεκτονικές, οι οποίες αποτελούν βελτιώσεις της LeNet, αλλά στηρίζονται κατά βάση στην εργασία του Yann LeCun. Η βασική δομή του CNN φαίνεται παρακάτω:



Εικόνα 5.1: Βασική δομή ενός Convolutional Neural Network [28]

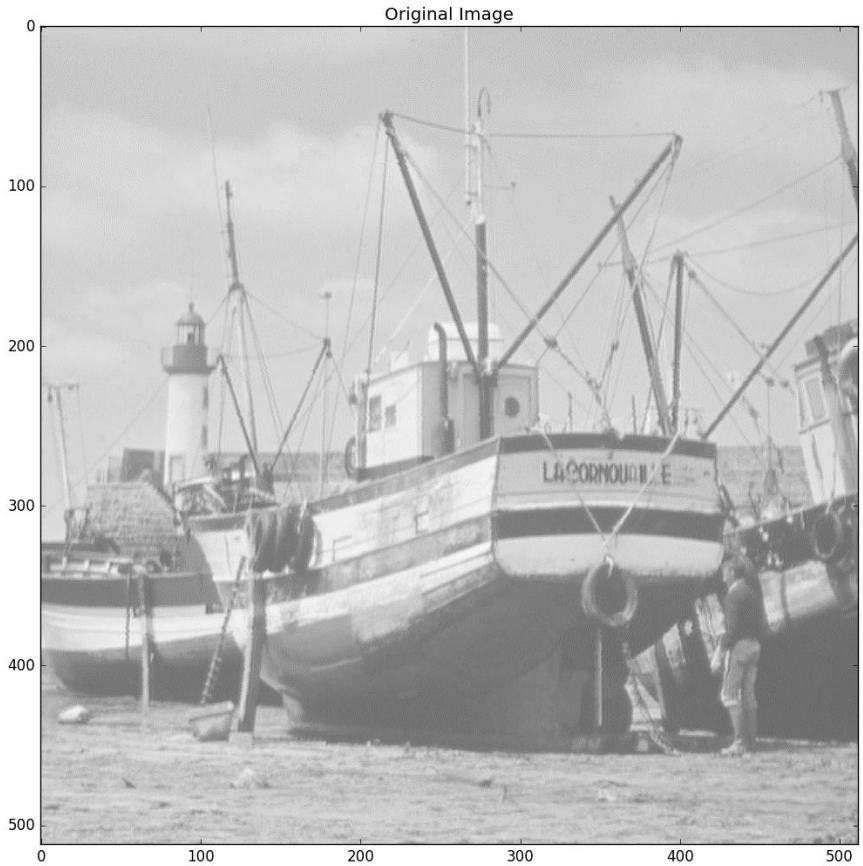
Το παραπάνω CNN επιδιώκει τη σωστή ταξινόμηση της εικόνας εισόδου σε μία από τις τέσσερις κατηγορίες οι οποίες είναι σκύλος, γάτα, σκάφος και πουλί. Έτσι, δίνοντας ως είσοδο μια εικόνα με δύο σκάφη, το νευρωνικό δίκτυο την ταξινομεί σωστά αποδίδοντάς της τη μεγαλύτερη πιθανότητα (0.94) σε σχέση με τις υπόλοιπες. Το άθροισμα των πιθανοτήτων είναι ίσο με τη μονάδα καθώς χρησιμοποιείται η συνάρτηση softmax στο output layer, όπως έχουμε δει ήδη στο προηγούμενο κεφάλαιο.

Υπάρχουν τέσσερις βασικές λειτουργίες σε κάθε CNN, και είναι οι ακόλουθες:

- Συνέλιξη
- Με γραμμικότητα (ReLU)
- Pooling ή υποδειγματοληψία
- Ταξινόμηση (fully connected layers)

Είδαμε στην προηγούμενη παράγραφο τον ορισμό της συνέλιξης (convolution), και πως αναμένουμε να τη χρησιμοποιήσουμε στα πλαίσια ενός αλγορίθμου εκμάθησης. Αναφέραμε ότι η είσοδος μας θα έχει το ρόλο της μίας συνάρτησης και κάποιο φίλτρο ή πυρήνας, που θα διαμορφώνεται από τον αλγόριθμο, θα έχει το ρόλο της δεύτερης. Τι επίδραση, όμως, έχει ένα φίλτρο σε μια εικόνα και σε τι μπορεί αυτό να μας φανεί χρήσιμο αν θέλουμε να την ταξινομήσουμε;

Ξεκινάμε, βλέποντας κάποια παραδείγματα. Θα πάρουμε σαν εικόνα αναφοράς αυτή που φαίνεται παρακάτω και θα εφαρμόσουμε συνέλιξη με δύο γνωστά φίλτρα:



Εικόνα 5.2: Αρχική εικόνα στην οποία θα εφαρμοστεί συνέλιξη με φίλτρα

Όπως φαίνεται και από τους άξονες, πρόκειται για μια εικόνα $512\text{x}512\ pixels$. Αρχικά, θα εφαρμόσουμε ένα γνωστό φίλτρο για sharpening, διαστάσεων $3\text{x}3$:

$$sharpen\ filter = \begin{bmatrix} 0 & -1 & 0 \\ -1 & +5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

Μέσω της συνέλιξης, κάθε στοιχείο του πίνακα (pixel) λαμβάνει μια νέα τιμή με βάση τους συντελεστές του φίλτρου πολλαπλασιασμένους με τις τιμές των γειτονικών στοιχείων, στο παράθυρο $3\text{x}3$ εδώ. Το αποτέλεσμα της συνέλιξης φαίνεται στην παρακάτω εικόνα:

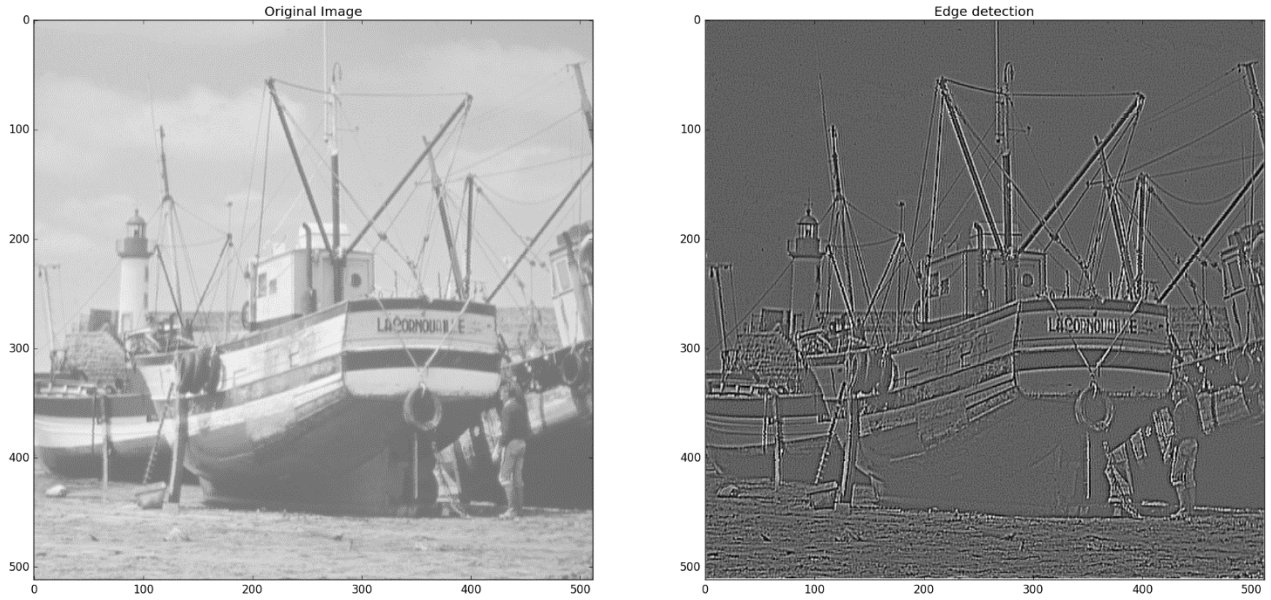
Εικόνα 5.3: Σύγκριση αρχικής εικόνας με αυτή που προκύπτει μετά τη συνέλιξη με sharpen φίλτρο

Παρατηρούμε ότι η εικόνα διαφοροποιήθηκε αρκετά καθώς τονίζονται πλέον στοιχεία που πριν δεν ήταν τόσο εμφανή.

Ας εφαρμόσουμε τώρα ένα άλλο γνωστό $3\text{x}3$ φίλτρο για ανίχνευση ακμών (edge detection), το οποίο είναι το ακόλουθο:

$$edge\ detection\ filter = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Παρακάτω παρουσιάζεται η διαφοροποίηση που επιφέρει το φίλτρο αυτό στην αρχική εικόνα και ο τονισμός των ακμών που υπάρχουν σε αυτή:



Εικόνα 5.4: Ανίχνευση ακμών στην αρχική εικόνα

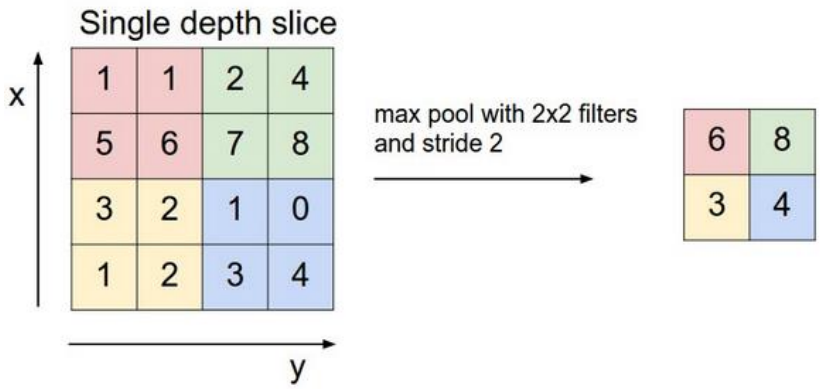
Αυτό που παρατηρούμε είναι ότι η συνέλιξη της αρχικής εικόνας με κάποιο φίλτρο (πυρήνας) επιφέρει μεγάλη διαφοροποίηση σε αυτή. Μετατρέπει, με άλλα λόγια, τη πληροφορία που είναι διαθέσιμη. Μήπως μπορούμε να επωφεληθούμε αρκετά από αυτό; Πώς θα μπορούσε να αξιοποιηθεί στα πλαίσια της μηχανικής μάθησης;

Παρέχουμε τη βασική αρχιτεκτονική, όπως το πλήθος των φίλτρων και τη διάσταση του καθενός, και αφήνουμε τον αλγόριθμο να προσδιορίσει ο ίδιος τις τιμές των παραμέτρων τους, μέσω της εκπαίδευσης. Αντί να ορίζουμε εξαρχής φίλτρα που θεωρούμε ότι θα συμβάλλουν στην αναγνώριση μιας εικόνας και στη σωστή ταξινόμησή της, διαμορφώνουμε το πλαίσιο ώστε αυτά να άυτο-προσδιοριστούν με κριτήριο την ελαχιστοποίηση της συνάρτησης κόστους. Αυτή είναι η βασική ιδέα πίσω από τα Convolutional neural networks. Αναφερόμαστε συχνά στη διαδικασία προσδιορισμού των φίλτρων, στα πρώτα layers ενός CNN, με τον όρο feature detection (ανίχνευση χαρακτηριστικών). Ποια χαρακτηριστικά, όμως, να ανιχνεύσουμε από μια εικόνα; Εκείνα που μας είναι χρήσιμα για την επίτευξη του σκοπού για το οποίο έχει δημιουργηθεί το μοντέλο, όπως τη σωστή ταξινόμηση των εικόνων.

Επιστρέφοντας στην αρχιτεκτονική του δικτύου μας, βλέπουμε ότι κάθε convolutional layer ακολουθείται από rectified linear units. Τα χαρακτηριστικά των ReLUs έχουν αναφερθεί στην παράγραφο 2.6 της εργασίας. Έστω ότι δίναμε μια εικόνα στην είσοδο με τιμές από 0 έως 255 σε κάθε pixel και βάθος 1 (κλίμακα του γκρι). Μετά τη συνέλιξη με τα διάφορα φίλτρα του πρώτου στρώματος, θα έχουν προκύψει θετικές και αρνητικές τιμές στα pixels των νέων εικόνων. Περνώντας τις εικόνες αυτές από τις ReLUs, κρατάμε ουσιαστικά μόνο τις θετικές τιμές κάνοντας όλες τις υπόλοιπες μηδέν.

Στις εικόνες που προκύπτουν από τις ReLUs, οι οποίες ονομάζονται και feature maps, εφαρμόζουμε pooling ή αλλιώς υποδειγματοληψία. Με τον τρόπο αυτό μειώνουμε τη διάσταση κάθε feature map διατηρώντας παράλληλα τις σημαντικότερες πληροφορίες. Υπάρχουν διάφοροι τρόποι να το επιτύχουμε αυτό, αλλά ο πιο συχνός είναι το max pooling κατά το οποίο ορίζουμε μια γειτονιά από στοιχεία (για παράδειγμα ένα

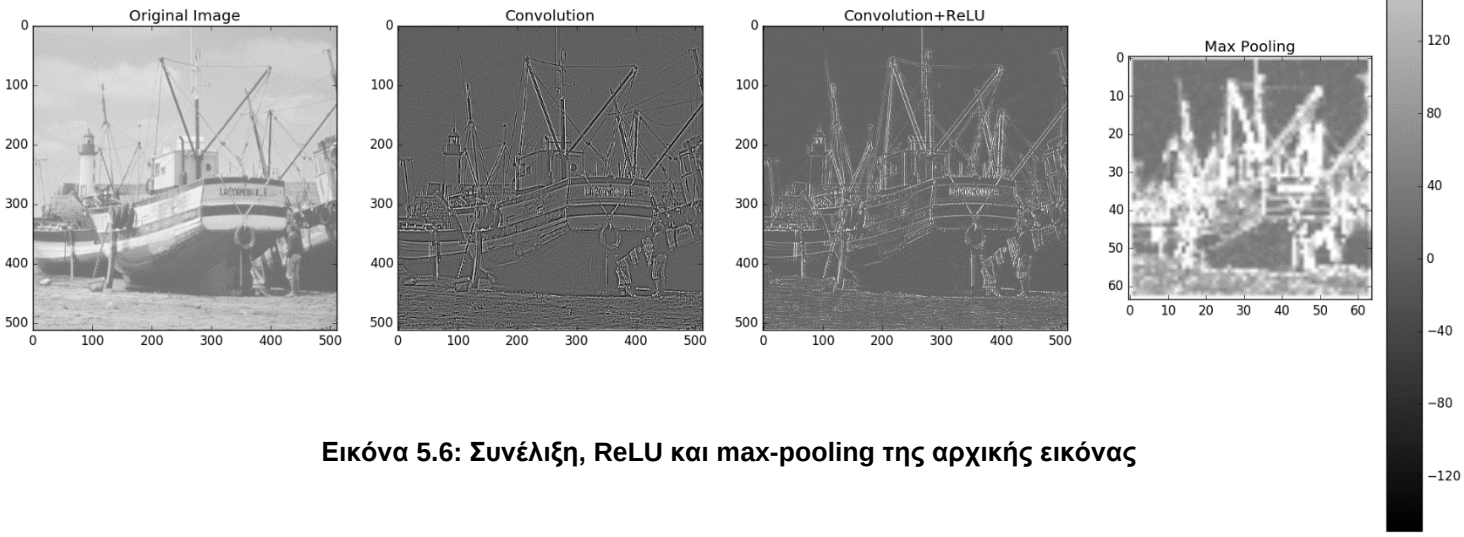
παράθυρο $2x2$) και κρατάμε τη μεγαλύτερη τιμή από τα στοιχεία της γειτονιάς αυτής. Ορίζουμε, επίσης, το βήμα (stride) που θα κάνουμε για να μεταβούμε από τη μία γειτονιά στην επόμενη. Στην εικόνα φαίνεται ένα παράδειγμα της διαδικασίας max pooling:



Εικόνα 5.5: Max pooling $2x2$ με stride 2 [29]

Ας δούμε τις λειτουργίες αυτές στην εικόνα που έχουμε σαν αντικείμενο. Υλοποιούμε συνέλιξη με το φίλτρο ανίχνευσης ακμών και περνάμε το αποτέλεσμα από ReLU. Στη συνέχεια εφαρμόζουμε max-pooling $8x8$ με βήμα 8 τόσο κάθετα όσο και οριζόντια υποοχταπλασιάζοντας έτσι και τις δύο διαστάσεις της.

Στην εικόνα παρακάτω, παρουσιάζονται μαζί τα βήματα που αναλύσαμε, τα οποία καταλήγουν στην υποδειγματοληψία του feature map μέσω του max-pooling:



Εικόνα 5.6: Συνέλιξη, ReLU και max-pooling της αρχικής εικόνας

Παρατηρούμε τα στάδια από τα οποία περνάει η αρχική εικόνα μας, έχοντας υπόψη την κλίμακα που έχει χρησιμοποιηθεί για την απεικόνισή τους. Μπορούμε να διαπιστώσουμε ότι οι αρνητικές τιμές (σκούρο γκρι και μαύρο χρώμα) μετά τη συνέλιξη, απαλείφονται στο στάδιο των ReLU. Ακόμα, η $512x512\ pixels$ εικόνα μειώνεται τελικά σε $64x64\ pixels$ έχοντας κρατήσει τη βασική πληροφορία από τη διαδικασία, δηλαδή την ανίχνευση των ακμών. Η τελική εικόνα είναι 64 φορές μικρότερη από το feature map

που προκύπτει από τις ReLUs, αλλά παρουσιάζεται σε παρόμοιο μέγεθος για να είναι ορατή στη λεπτομέρειά της.

Η πληροφορία μετά το max-pooling αναπαριστά υψηλού-επιπέδου χαρακτηριστικά της αρχικής εικόνας. Αυτά καταλήγουν σε ένα κλασικό πολυστρωματικό perceptron (fully connected layers), το οποίο συνήθως χρησιμοποιεί softmax συνάρτηση ενεργοποίησης στο στρώμα εξόδου για το classification. Σκοπός των fully connected layers είναι να χρησιμοποιήσουν τα features αυτά για να ταξινομήσουν σωστά την εικόνα.

Πέρα από την ταξινόμηση, προσθέτοντας ένα fully-connected layer επιτυγχάνουμε την εκμάθηση μη γραμμικών συνδυασμών των χαρακτηριστικών που έχουν εξαχθεί, για την σωστότερη ταξινόμηση των εικόνων εισόδου. Όπως έχει αναφερθεί, με τη συνάρτηση softmax θα έχουμε τις πιθανότητες να ανήκει η εικόνα εισόδου σε κάθε κατηγορία, οι οποίες θα αθροίζουν στη μονάδα. [30]

5.3 Υλοποίηση CNN με Dropout στο TensorFlow για την αναγνώριση ψηφίων

Πρόκειται να υλοποιήσουμε ένα Convolutional Neural Network που θα δέχεται στην είσοδο την εικόνα ενός ψηφίου $28\text{x}28\,pixels$, θα την μεταβιβάζει στο πρώτο convolutional layer με ReLU το οποίο θα αποτελείται από 32 φίλτρα. Στη συνέχεια, θα υπάρχει το πρώτο pooling layer, που θα υλοποιεί max-pooling $2\text{x}2$. Η πληροφορία θα μεταβιβάζεται μετά στο δεύτερο convolutional layer με ReLU αποτελούμενο από 64 φίλτρα. Ένα δεύτερο pooling layer ακολουθεί με νέο max-pooling $2\text{x}2$ το οποίο παραδίνει την πληροφορία σε ένα fully connected layer αποτελούμενο από 1000 νευρώνες. Το layer αυτό συνδέεται πλήρως με τους 10 νευρώνες του στρώματος εξόδου το οποίο με τη χρήση της softmax εξάγει τις πιθανότητες να ανήκει η εικόνα σε κάθε κατηγορία και την ταξινομεί στην πιο πιθανή.

Ξεκινάμε, όπως και στην περίπτωση του feed forward δικτύου, με τον ορισμό μεθόδων που θα χρησιμοποιήσουμε στο μοντέλο και με την ανάγνωση των δεδομένων εισόδου:

```
import tensorflow as tf
import numpy as np
from tensorflow.examples.tutorials.mnist import input_data

def init_weights(shape):
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def init_biases(shape):  # slightly positive initial bias to avoid "dead" neurons
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# convolution with a stride of one and zero padding
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

# max pooling over 2x2 blocks
def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

```
def out_layer(h, w_o, b_o):
    return (tf.matmul(h, w_o) + b_o) # note that we dont take the softmax at the
end because our cost includes it

# input data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
trX, trY, teX, teY = mnist.train.images, mnist.train.labels, mnist.test.images,
mnist.test.labels
```

Βλέπουμε ότι αξιοποιούμε τις μεθόδους tf.nn.conv2d() και tf.nn.max_pool() περνώντας τους τις κατάλληλες παραμέτρους [25]. Συνεχίζουμε με τον προσδιορισμό των υπερ-παραμέτρων του δικτύου μας και τη δήλωση των placeholders που θα χρησιμοποιήσουμε:

```
# define hyper-parameters' values
dropout_hidden = 0.5
mini_batch_size = 100
learning_rate = 0.01
epochs = 60

X = tf.placeholder("float", [None, 784])
Y = tf.placeholder("float", [None, 10])
keep_prob_hidden = tf.placeholder(tf.float32)
```

Θα χρησιμοποιήσουμε dropout και εδώ, συγκεκριμένα στο fully connected layer. Δεν θα είχε νόημα να επέμβουμε με dropout στη διαδικασίας εξαγωγής των χρήσιμων χαρακτηριστικών της εικόνας, για αυτό και δεν χρησιμοποιείται στα convolutional και στα pooling layers.

Συνεχίζουμε ορίζοντας τα βάρη για το μοντέλο μας. Όπως αναφέραμε στο 1ο convolutional layer θα έχουμε 32 φίλτρα. Επιλέγουμε αυτά να είναι διάστασης 5x5 το καθένα. Στην είσοδό τους δέχονται 1 channel καθώς πρόκειται για grayscale εικόνα:

```
# The first two dimensions are the patch size {5x5 px}
# the next is the number of input channels,
# and the last is the number of output channels
# So 32 filters with size 5x5 each
w_conv1 = init_weights([5, 5, 1, 32])
b_conv1 = init_biases([32])
```

Στο 2ο convolutional layer θα έχουμε 64 φίλτρα, και πάλι 5x5, που θα δέχονται στην είσοδό τους 32 channels, όσα εξάγονται δηλαδή από το προηγούμενο layer:

```
w_conv2 = init_weights([5, 5, 32, 64])
b_conv2 = init_biases([64])
```

Η εικόνα μας στο σημείο αυτό έχει περάσει το 1ο pooling layer, με max-pooling 2x2 και stride 2 και στις δύο διαστάσεις, όπου από 28x28 έγινε 14x14 καθώς και το 2ο pooling layer με τα ίδια χαρακτηριστικά στο οποίο μειώθηκε η διάστασή της σε 7x7. Ορίζουμε, λοιπόν, για το fully connected καθώς και για το output layer:

```
# fully connected layer, 7x7 the reduced image size after two max-pooling
operations
w_fc1 = init_weights([7 * 7 * 64, 1000])
b_fc1 = init_biases([1000])

# output layer
w_o = init_weights([1000, 10])
b_o = init_biases([10])
```

Αφού έχουμε δηλώσει όλα τα symbolic variables, προχωράμε στον καθορισμό του μοντέλου:

```
# define the model

# input tensor must have 4 dimensions: [batch, height, width, channels],
# -1 in order to adjust as necessary to match the size needed for the full tensor
X_as_image = tf.reshape(X, [-1, 28, 28, 1])

# first convolutional layer
h_conv1 = tf.nn.relu(conv2d(X_as_image, w_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# second convolutional layer
h_conv2 = tf.nn.relu(conv2d(h_pool1, w_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# reshape output for the first fully-connected layer
h_pool2_flat = tf.reshape(h_pool2, [-1, 7 * 7 * 64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, w_fc1) + b_fc1)

# use dropout in the fully connected layer
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob_hidden)

# output layer
y_bef_softmax = out_layer(h_fc1_drop, w_o, b_o)
```

Ορίζουμε, ακόμα, τη συνάρτηση κόστους και τον αλγόριθμο βαελτιστοποίησης:

```
# define cost function and optimization algorithm
cost =
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y_bef_softmax,
labels=Y))
train_op = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
predict_op = tf.argmax(y_bef_softmax, 1)
```

Τέλος, δημιουργούμε το session object, αρχικοποιούμε τις μεταβλητές και υπολογίζουμε τα tensors στο μοντέλο μας, περνώντας του κάθε φορά τα κατάλληλα δεδομένα:

```
# Launch the graph in a session
sess = tf.Session()
```

```
# initialize all variables
sess.run(tf.global_variables_initializer())

for i in range(epochs):
    for start, end in zip(range(0, len(trX), mini_batch_size),
range(mini_batch_size, len(trX)+1, mini_batch_size)):
        sess.run(train_op, feed_dict={X: trX[start:end], Y: trY[start:end],
keep_prob_hidden: (1.0 - dropout_hidden)})

    wrong_classified = np.sum(np.argmax(teY, axis=1) != sess.run(predict_op,
feed_dict={X: teX, keep_prob_hidden: 1.0}))
    print(i, wrong_classified)
```

Στην έξοδο τυπώνει τον αριθμό των λανθασμένα ταξινομημένων εικόνων από τις 10000 του test set στο τέλος κάθε εποχής.

5.4 Πειραματικά αποτελέσματα της χρήσης CNN για την αναγνώριση ψηφίων

Υλοποιήσαμε, λοιπόν, το CNN με την ακόλουθη αρχιτεκτονική:

- **Εικόνα εισόδου:** $28\text{x}28\ px$

- **Convolutional layer 1 + ReLU:** 32 φίλτρα $5\text{x}5$ το καθένα και 1 bias για κάθε φίλτρο. Χρησιμοποιήσαμε zero-padding (γέμισμα με μηδενικά γύρω από τα όρια), έτσι ώστε η εικόνα εξόδου να έχει ίδια διάσταση με την εικόνα εισόδου, $28\text{x}28\ px$. Στη συνέχεια εφαρμόσαμε ReLU.

- **Pooling layer 1:** Max Pooling σε κάθε μπλοκ $2\text{x}2\ px$. Το αποτέλεσμα είναι 32 κανάλια (από τα προηγούμενα 32 φίλτρα) διάστασης $14\text{x}14\ px$ το καθένα.

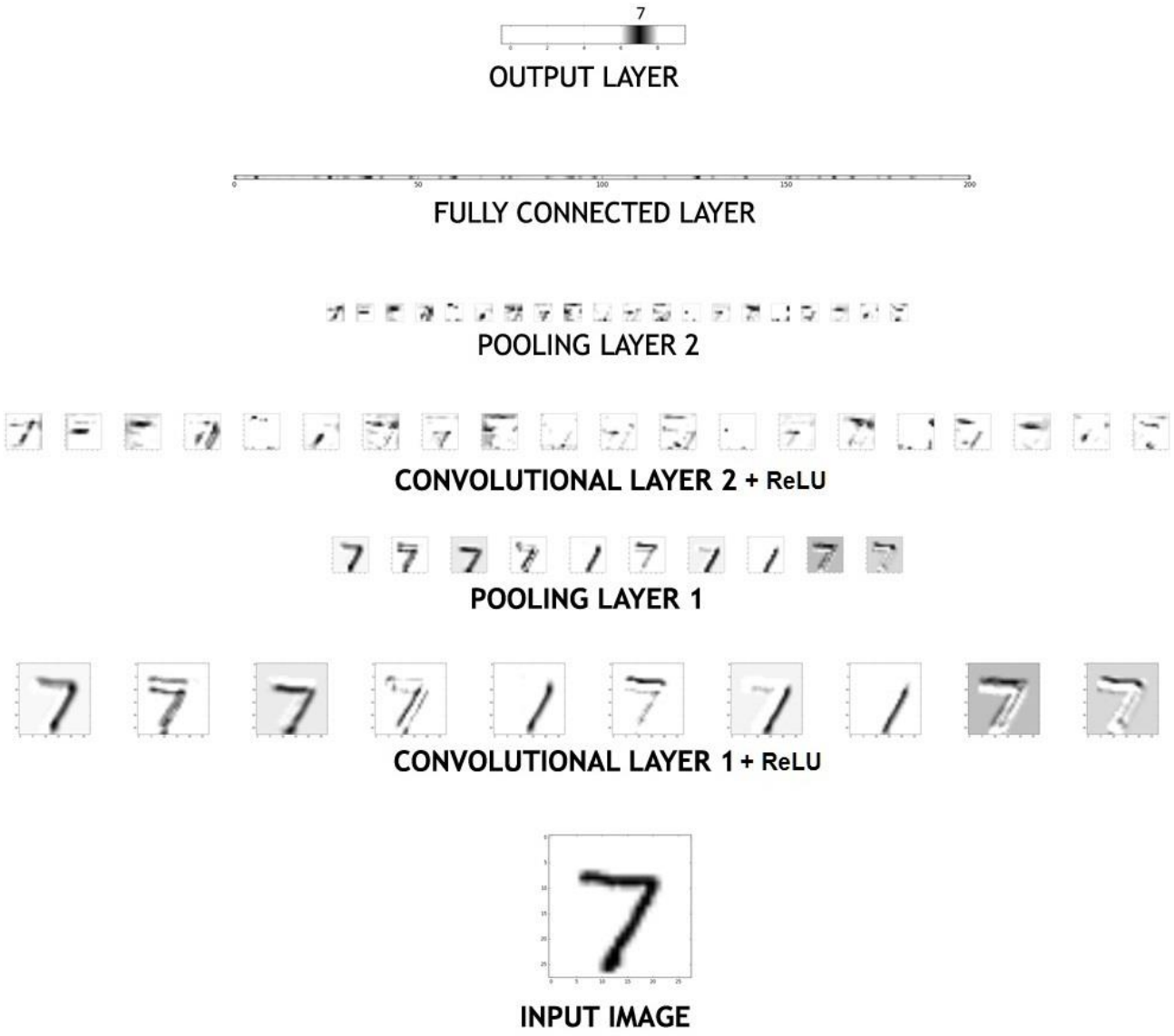
- **Convolutional layer 2 + ReLU:** 64 φίλτρα $5\text{x}5$ το καθένα και 1 bias για κάθε φίλτρο. Χρησιμοποιήσαμε zero-padding και εδώ. Το αποτέλεσμα είναι 64 παραγόμενες εικόνες, $14\text{x}14\ px$ η κάθε μία. Στη συνέχεια περάσανε από ReLU.

- **Pooling layer 2:** Max Pooling σε κάθε μπλοκ $2\text{x}2\ px$. Άρα, έχουμε 64 κανάλια των $7\text{x}7\ px$.

- **Fully Connected layer:** 1000 νευρώνες με rectifier για συνάρτηση ενεργοποίησης. Δηλαδή, στον κάθε νευρώνα καταλήγουν $7 * 7 * 64$ βάρη σε αυτό το στρώμα. Ο συνολικός αριθμός των βαρών του στρώματος είναι $7 * 7 * 64 * 1000$.

- **Output layer:** 10 νευρώνες, πλήρως διασυνδεδεμένοι με το προηγούμενο στρώμα, με softmax συνάρτηση ενεργοποίησης για το classification.

Στην παρακάτω εικόνα μπορούμε να δούμε όλα τα στρώματα του εκπαιδευμένου CNN, με κάποια δειγματοληψία, με είσοδο ένα ψηφίο γραμμένο με το χέρι:



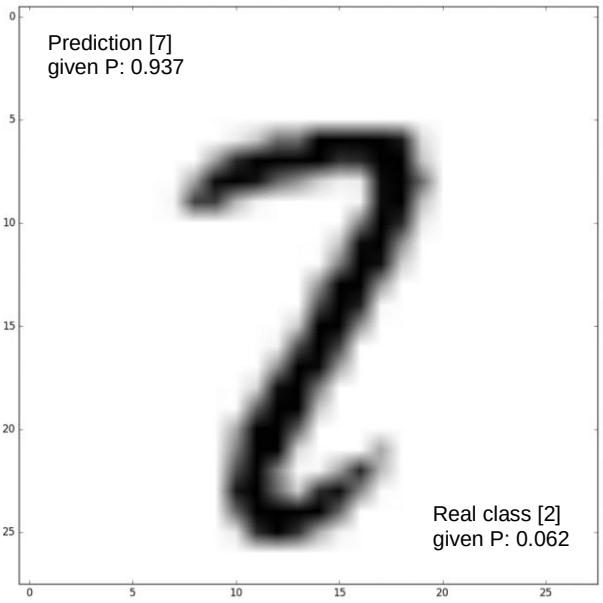
Εικόνα 5.7: Εποπτική παρουσίαση όλων των στρωμάτων του CNN με είσοδο ένα ψηφίο

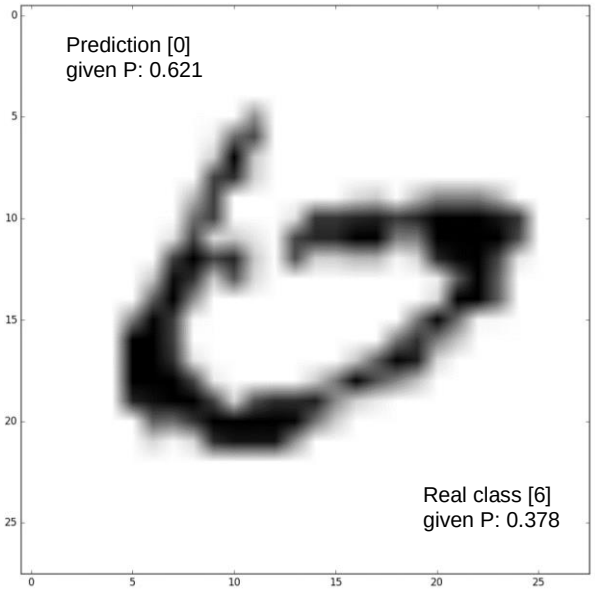
Παραπάνω, βλέπουμε αρκετά καθαρά τα στάδια από τα οποία περνάει η αρχική πληροφορία, πως μετατρέπεται, ώστε να κατηγοριοποιηθεί εν τέλει σωστά στο στρώμα εξόδου.

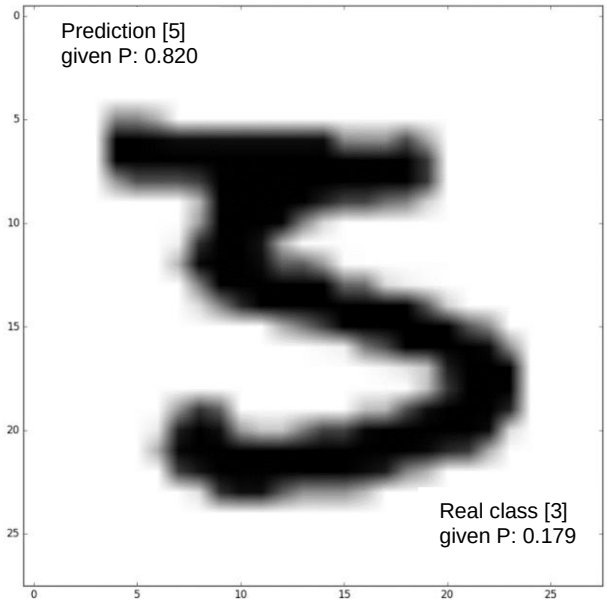
Ασφαλώς, στην εικόνα υπάρχει ένα μέρος από το σύνολο των φίλτρων στα convolutional layers και των καναλιών των pooling layers, ώστε να μπορούν αυτά να είναι εμφανή.

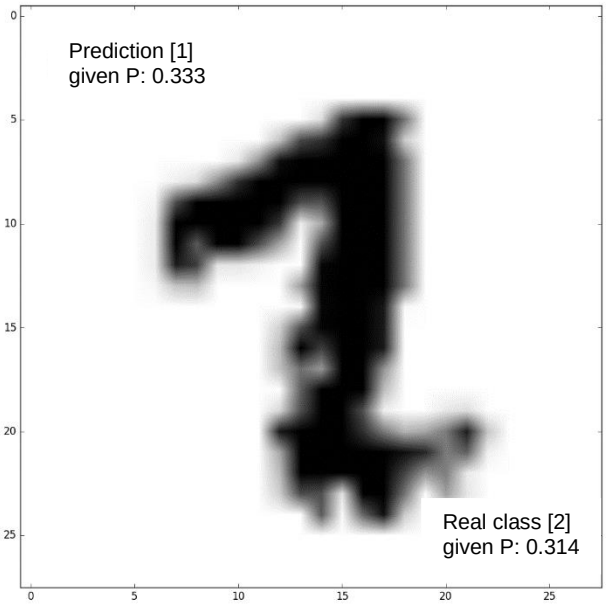
Για να μιλήσουμε με περισσότερη ακρίβεια, αφού θυμηθούμε ότι η καλύτερη απόδοση στα feedforward NNs ήταν 98.84%, να αναφέρουμε ότι εδώ ανεβήκαμε στο $\mathbf{99.22}\%$, με χρήση Dropout 50% στο fully connected layer.

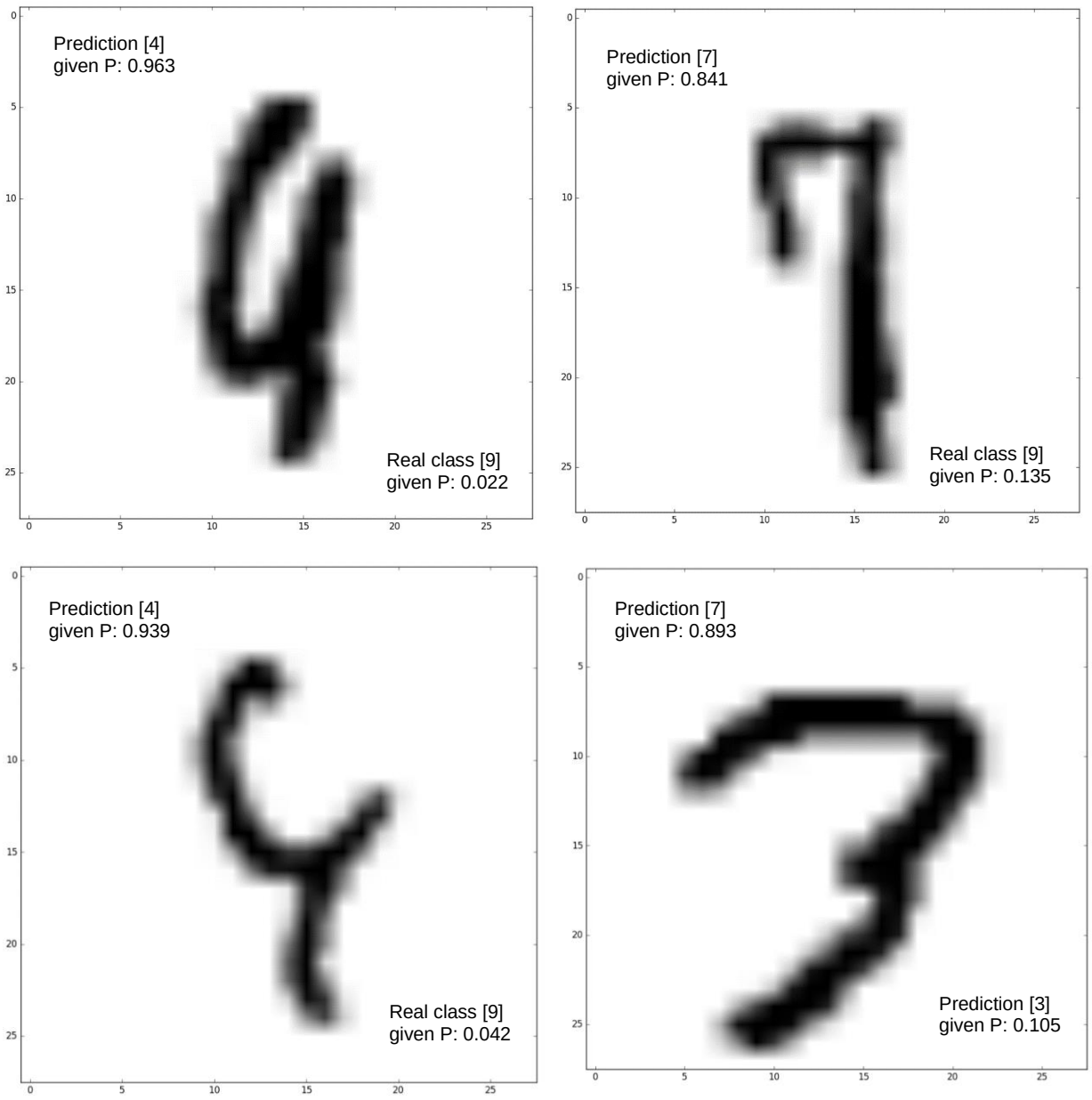
Αυτό σημαίνει ότι από τα 10000 ψηφία του συνόλου ελέγχου, μόνο τα 78 δεν ταξινομούνται στη σωστή κατηγορία από το CNN. Ας δούμε τώρα κάποιες περιπτώσεις τέτοιων ψηφίων:









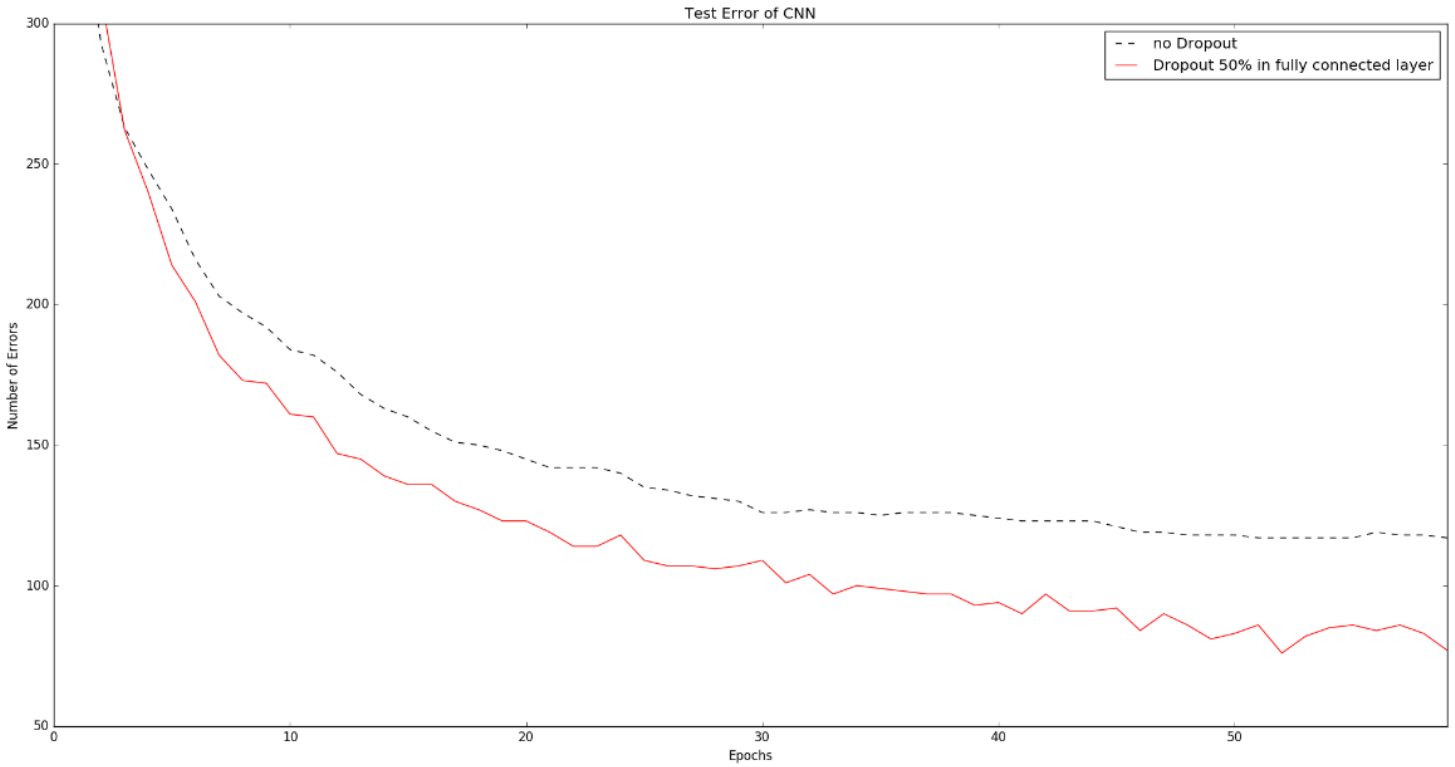


Εικόνα 5.8: Λανθασμένα κατηγοριοποιημένα ψηφία από το CNN με τις δοθείσες πιθανότητες

Στις εικόνες των ψηφίων παραπάνω φαίνονται τόσο η πιθανότητα που δόθηκε στην κατηγορία που λανθασμένα ταξινομήθηκε το ψηφίο, επάνω αριστερά, όσο και η πιθανότητα που δόθηκε στη σωστή κατηγορία, κάτω δεξιά. Για παράδειγμα, το πρώτο ψηφίο, επάνω αριστερά, αναγνωρίστηκε λανθασμένα ως 7 με πιθανότητα 93.7% ενώ στην πραγματική του κατηγορία, 2, δόθηκε πιθανότητα 6.2% από το μοντέλο. Αντίστοιχα, βλέπουμε και για άλλα ψηφία, από τα 78 που ταξινομήθηκαν εσφαλμένα, την έξοδο του μοντέλου, κατανοώντας με αυτό τον τρόπο περισσότερο τη λειτουργία του.

Όπως και στο προηγούμενο κεφάλαιο, ελέγξαμε και εδώ τη συμβολή του dropout στη βελτίωση της απόδοσης του αλγορίθμου. Είδαμε ότι dropout εδώ εφαρμόζεται στο fully connected layer, ίσο με 50%. Τρέξαμε την εκμάθηση του μοντέλου για 60 εποχές, τη μία φορά με χρήση dropout και την άλλη όχι, διατηρώντας όλες τις υπόλοιπες παραμέτρους

του προβλήματος ίδιες. Στο γράφημα, φαίνεται το πλήθος των λανθασμένα ταξινομημένων ψηφίων για κάθε εποχή, στις δύο περιπτώσεις:



Σχήμα 5.1: Σύγκριση της απόδοσης του Convolutional Neural Network με και χωρίς Dropout

Παρατηρούμε και εδώ ότι η χρήση του dropout για regularization ενίσχυσε σημαντικά την απόδοση του αλγορίθμου. Συγκεκριμένα, η ακρίβεια του μοντέλου χωρίς dropout ήταν 98.83%, ενώ με τη χρήση dropout ανέβηκε στο $\mathbf{99.22}\%$.

6. ΜΕΛΕΤΗ ΤΩΝ RECURRENT NEURAL NETWORKS ΜΕ ΧΡΗΣΗ DROPOUT

6.1 Εισαγωγή στα Recurrent Neural Networks

Τα Recurrent Neural Networks ή RNNs είναι μια οικογένεια νευρωνικών δικτύων ιδιαίτερα ικανών στην επεξεργασία δεδομένων που παρατηρούνται ως ακολουθία (sequential data). Όπως τα CNNs είναι ιδανικά στην επεξεργασία εικόνων, τα RNNs ειδικεύονται στην επεξεργασία μια ακολουθίας τιμών $x^{(1)}, \dots, x^{(\tau)}$.

Για να περάσουμε από τα πολυστρωματικά νευρωνικά δίκτυα στα RNNs, θα πρέπει να εκμεταλλευτούμε μια βασική ιδέα που εμφανίστηκε περί το 1980 στη μηχανική μάθηση και τη στατιστική μοντελοποίηση, αυτή της κοινής χρήσης των παραμέτρων (parameters' sharing) σε διαφορετικά σημεία του μοντέλου. Η κοινή χρήση των παραμέτρων είναι αυτή που μας δίνει τη δυνατότητα να επεκτείνουμε και να εφαρμόσουμε το μοντέλο σε διαφορετικές μορφές (διαφορετικά μήκη εδώ) και να γενικεύσουμε σε αυτές. Αν είχαμε ξεχωριστές παραμέτρους για κάθε τιμή του χρονικού βήματος, δεν θα μπορούσαμε να γενικεύσουμε σε μήκη ακολουθιών που δεν έχουν παρουσιαστεί στην εκπαίδευση, ούτε να εκμεταλλευτούμε τη στατιστική γνώση σε ακολουθίες διαφορετικών μηκών ή χρονικά μετατοπισμένων.

Μια τέτοιου είδους εκπαίδευση είναι απαραίτητη όταν μια πληροφορία μπορεί να παρατηρηθεί σε διαφορετικές θέσεις εντός της ακολουθίας. Για παράδειγμα, ας υποθέσουμε ότι έχουμε τις προτάσεις «Πήγα στο Νεπάλ το 2009» και «Το 2009, πήγα στο Νεπάλ». Αν ζητήσουμε από ένα μοντέλο μηχανικής μάθησης, στο οποίο θα δώσουμε κάθε πρόταση, να εξάγει τη χρονιά στην οποία ο αφηγητής πήγε στο Νεπάλ, θέλουμε να αναγνωρίζει το έτος 2009 ως τη σχετιζόμενη με την ερώτηση πληροφορία, είτε βρίσκεται στην αρχή είτε στο τέλος της πρότασης. Έστω ότι εκπαιδεύαμε ένα feedforward δίκτυο που επεξεργάζεται προτάσεις ορισμένου μήκους. Ένα παραδοσιακό πλήρως διασυνδεδεμένο νευρωνικό δίκτυο θα είχε διαφορετικές παραμέτρους για κάθε χαρακτηριστικό εισόδου και συνεπώς θα έπρεπε να μάθει όλους τους κανόνες της γλώσσας χωριστά για κάθε διαφορετική θέση ενός στοιχείου στην πρόταση. Σε αντίθεση με αυτό, ένα RNN έχει τη δυνατότητα της κοινής χρήση των βαρών του σε διαφορετικά χρονικά βήματα.

Σε ένα RNN, κάθε στοιχείο εξόδου είναι συνάρτηση των προηγούμενων (χρονικά) στοιχείων εξόδου. Τα στοιχεία εξόδου παράγονται υλοποιώντας τον ίδιο κανόνα ενημέρωσης όπως και στα προηγούμενα από αυτά. Αυτός ο επαναλαμβανόμενος (recurrent) σχηματισμός είναι που επιτρέπει την κοινή χρήση των παραμέτρων μέσα σε ένα βαθύ υπολογιστικό γράφο.

Για λόγους απλότητας, αναφέρουμε ότι τα RNNs δρουν σε μία ακολουθία από διανύσματα $\boldsymbol{x}^t$ όπου το χρονικό βήμα t κυμαίνεται από 1 έως τ. Στην πράξη, τα RNNs δέχονται συνήθως μικρά σύνολα (minibatches) τέτοιων ακολουθιών όπου κάθε μέλος του minibatch μπορεί να έχει διαφορετικό μήκος ακολουθίας τ. Επίσης, το χρονικό βήμα δεν υποδηλώνει κατά κυριολεξία το ρεαλιστικό πέρασμα του χρόνου. Πολλές φορές, για παράδειγμα, δείχνει τη θέση στην ακολουθία.

Ας μελετήσουμε πιο προσεκτικά τον υπολογιστικό γράφο που υλοποιούν τα recurrent neural networks. Αρχικά, ο υπολογιστικός γράφος είναι μια μέθοδος αναπαράστασης της δομής των υπολογισμών, όπως αυτοί που υλοποιούνται στην αντιστοίχιση της εισόδου και των παραμέτρων με την έξοδο και τις απώλειες. Για παράδειγμα, μια κλασική μορφή δυναμικού συστήματος είναι η ακόλουθη:

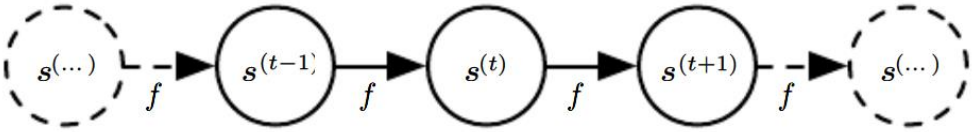
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}; \boldsymbol{\theta}) ,$$

όπου $\boldsymbol{s}^{(t)}$ είναι η κατάσταση του συστήματος.

Η εξίσωση αυτή είναι επαναλαμβανόμενη αφού ο ορισμός της κατάστασης s τη χρονική στιγμή t οδηγεί στον ίδιο ορισμό για τη χρονική στιγμή $t-1$. Για έναν πεπερασμένο αριθμό βημάτων τ, ο γράφος μπορεί να αναπτυχθεί εφαρμόζοντας τον ορισμό $\tau-1$ φορές. Έτσι, για $\tau=3$ χρονικά βήματα, θα έχουμε:

$$\boldsymbol{s}^{(3)} = f(\boldsymbol{s}^{(2)};\boldsymbol{\theta}) = f(f(\boldsymbol{s}^{(1)};\boldsymbol{\theta});\boldsymbol{\theta})$$

Μια τέτοια έκφραση μπορεί να αναπαρασταθεί από έναν κλασικό κατευθυνόμενο ακυκλικό γράφο:



Εικόνα 6.1: Ανεπτυγμένος υπολογιστικός γράφος κλασικού δυναμικού συστήματος

Κάθε κόμβος αναπαριστά την κατάσταση σε κάποιο χρονικό βήμα t και η συνάρτηση f αντιστοιχίζει την κατάσταση στο βήμα t με εκείνη στο βήμα $t+1$. Οι ίδιες παράμετροι, $\boldsymbol{\theta}$, χρησιμοποιούνται για όλα τα χρονικά βήματα. Σαν άλλο παράδειγμα, ας σκεφτούμε ένα δυναμικό σύστημα που οδηγείται από ένα εξωτερικό σήμα $\boldsymbol{x}^{(t)}$:

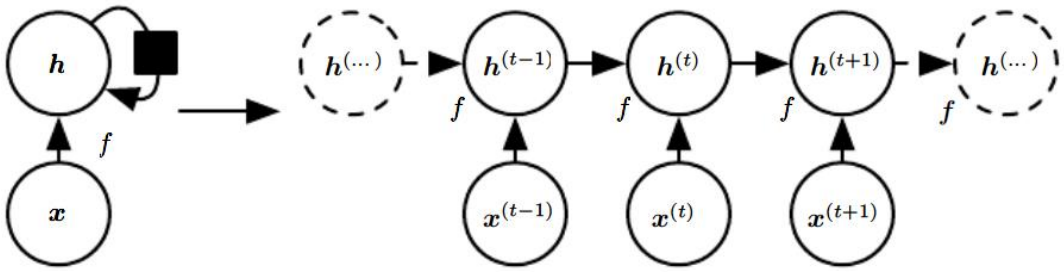
$$\boldsymbol{s}^{(t)} = f(\boldsymbol{s}^{(t-1)}, \boldsymbol{x}^{(t)};\boldsymbol{\theta}) \,,$$

στο οποίο βλέπουμε ότι η τρέχουσα κατάσταση περιέχει την πληροφορία για το σύνολο της ακολουθίας που έχει παρέλθει.

Τα RNNs μπορούν να αναπτυχθούν με πολλούς διαφορετικούς τρόπους. Όπως κάθε συνάρτηση μπορεί να λογιστεί ως ένα feedforward νευρωνικό δίκτυο, έτσι και κάθε συνάρτηση που εμπεριέχει επαναληψιμότητα μπορεί να λογιστεί ως ένα recurrent νευρωνικό δίκτυο. Πολλά RNNs χρησιμοποιούν την ακόλουθη, ή μια παρόμοια με αυτή, εξίσωση για την κατάσταση $\boldsymbol{h}$ των hidden units του δικτύου:

$$\boldsymbol{h}^{(t)} = f(\boldsymbol{h}^{(t-1)}, \boldsymbol{x}^{(t)};\boldsymbol{\theta}) \,.$$

Αναπαριστούμε γραφικά ένα τέτοιο δίκτυο, χωρίς στρώμα εξόδου, το οποίο επεξεργάζεται και μαθαίνει συνδυάζοντας την πληροφορία της εισόδου $\boldsymbol{x}$ με την κατάσταση $\boldsymbol{h}$, τροφοδοτώντας με την τελευταία τα επόμενα χρονικά βήματα:



Εικόνα 6.2: Κυκλωματικό διάγραμμα με καθυστέρηση και αντίστοιχος ανεπτυγμένος γράφος

Στην αρχιτεκτονική των RNNs θα υπάρχει άλλο ένα χαρακτηριστικό, τα στρώματα εξόδου που διαβάζουν την πληροφορία από την κατάσταση $\boldsymbol{h}$, ώστε το δίκτυο να κάνει εκτιμήσεις.

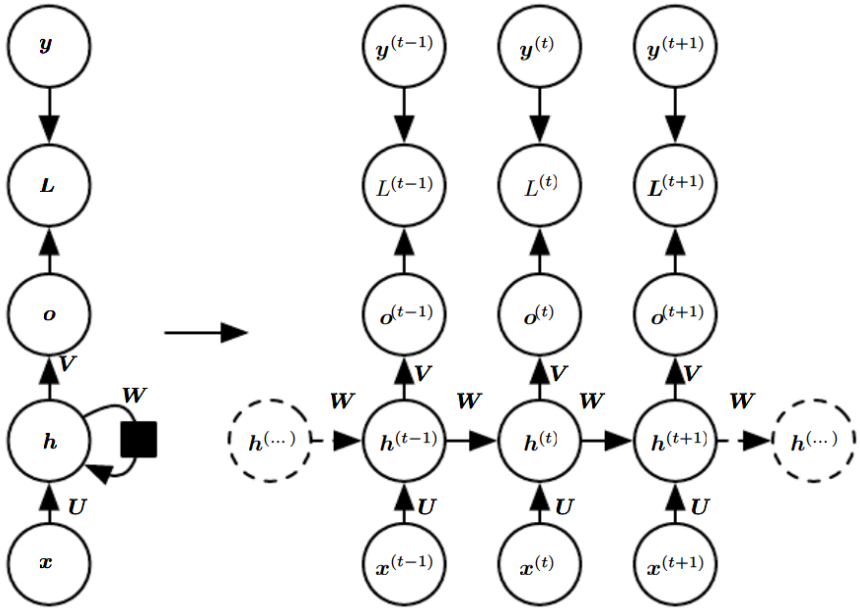
Καθώς το RNN εκπαιδεύεται για κάποιο σκοπό που απαιτεί την πρόβλεψη του μέλλοντος από το παρελθόν, αξιοποιεί τη γνώση μια τμηματικής σύνοψης των παρελθόντων καταστάσεων $\boldsymbol{h}^{(t)}$ μέχρι τη χρονική στιγμή t. Η σύνοψη αυτή είναι, εν γένει, τμηματική από τη στιγμή που αντιστοιχίζει μια αυθαίρετου μήκους ακολουθία $(\boldsymbol{x}^{(t)}, \boldsymbol{x}^{(t-1)}, \boldsymbol{x}^{(t-2)}, \dots, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(1)})$ σε ένα συγκεκριμένου μήκους διάνυσμα $\boldsymbol{h}^{(t)}$. Αναλόγως το κριτήριο εκπαίδευσης, η σύνοψη αυτή μπορεί, επιλεκτικά, να έχει κρατήσει κάποια χαρακτηριστικά του παρελθόντος και κάποια άλλα όχι. Για παράδειγμα ένα RNN που χρησιμοποιείται για τη στατιστική μοντελοποίηση γλώσσας, το οποίο κατά κανόνα προβλέπει την επόμενη λέξη έχοντας στη διάθεσή του τις προηγούμενες, πιθανότατα δεν χρειάζεται να διατηρεί στη μνήμη του όλη την πληροφορία της πρότασης μέχρι το βήμα t, αλλά μόνο τις παρελθούσες καταστάσεις που του είναι χρήσιμες για να προβλέψει το υπόλοιπο της πρότασης.

Τα πολύ σημαντικά πλεονεκτήματα που έχουν τα recurrent neural networks είναι ότι:

1. Ανεξάρτητα από το μήκος της ακολουθίας, το εκπαιδευόμενο μοντέλο ορίζεται μέσω της μετάβασης από τη μία κατάσταση στην άλλη, όχι από το μεταβλητό μήκος του διαθέσιμου ιστορικού των καταστάσεων, και διατηρεί για το λόγο αυτό το ίδιο μέγεθος στα δεδομένα εισόδου.
2. Έχουμε τη δυνατότητα να χρησιμοποιήσουμε μια κοινή συνάρτηση μετάβασης με τις ίδιες παραμέτρους σε κάθε χρονικό βήμα.

Αυτοί οι δύο παράγοντες κάνουν εφικτή την εκμάθηση ενός μοναδικού μοντέλου που λειτουργεί σε όλα τα χρονικά βήματα και όλα τα μήκη ακολουθιών, αντί να είναι αναγκαία η εκμάθηση διαφορετικού μοντέλου για όλα τα πιθανά χρονικά βήματα. Ακόμα, η εκπαίδευση ενός μοναδικού μοντέλου επιτρέπει τη γενίκευση σε μήκη ακολουθιών που δεν υπάρχουν στο σύνολο εκπαίδευσης και καθιστά δυνατή την εκμάθησή του με λιγότερα παραδείγματα εκπαίδευσης σε σχέση με εκείνα που θα απαιτούνταν αν δεν υπήρχε κοινή χρήση των παραμέτρων.

Υπάρχουν αρκετές αρχιτεκτονικές ανάπτυξης των RNNs, αλλά ας επικεντρωθούμε σε μία τυπική από αυτές:



Εικόνα 6.3: Τυπική αρχιτεκτονική Recurrent Neural Network

Αριστερά βλέπουμε το κυκλωματικό διάγραμμα και δεξιά τον χρονικά ανεπτυγμένο υπολογιστικό γράφο. Με το μαύρο τετράγωνο συμβολίζεται η χρονική καθυστέρηση ενός βήματος.

Σκοπός ενός τέτοιου μοντέλου είναι η αντιστοίχιση μιας ακολουθίας εισόδου με $\boldsymbol{x}$ τιμές, σε μία ακολουθία από $\boldsymbol{o}$ τιμές εξόδου. Το κριτήριο απωλειών L μετράει πόσο απέχει κάθε o από την αντίστοιχη ορθή τιμή y. Όταν χρησιμοποιούμε softmax στην έξοδο, θεωρούμε ότι $\boldsymbol{o}$ είναι οι μη κανονικοποιημένες πιθανότητες. Το κριτήριο απωλειών L υπολογίζει την εκτιμώμενη έξοδο $\hat{\boldsymbol{y}} = softmax(\boldsymbol{o})$ και τη συγκρίνει με τον στόχο $\boldsymbol{y}$.

Το RNN έχει διασυνδέσεις μεταξύ εισόδου και κρυφού στρώματος που παραμετροποιούνται από τον πίνακα βαρών $\boldsymbol{U}$, διασυνδέσεις κρυφού στρώματος σε κρυφό στρώμα παραμετροποιημένες από τον πίνακα $\boldsymbol{W}$ και διασυνδέσεις από το κρυφό στρώμα προς το στρώμα εξόδου, οι παράμετροι των οποίων συμβολίζονται με τον πίνακα $\boldsymbol{V}$.

Προχωράμε στο να διατυπώσουμε τις εξισώσεις που περιγράφουν την εμπρόσθια μετάδοση της πληροφορίας στο RNN της εικόνας 6.3. Υποθέτουμε ότι χρησιμοποιούμε την υπερβολική εφαπτομένη ως συνάρτηση ενεργοποίησης στους νευρώνες του κρυφού στρώματος. Θεωρούμε, ακόμα, ότι η έξοδός μας είναι διακριτή, όπως θα ήταν στην περίπτωση που το RNN προέβλεπε επόμενους χαρακτήρες ή λέξεις σε ένα πρόβλημα μοντελοποίησης γλώσσας. Ένας τυπικός τρόπος αναπαράστασης των διακριτών μεταβλητών είναι μέσω του διανύσματος εξόδου $\boldsymbol{o}$ εμπεριέχοντας σε αυτό τις μη κανονικοποιημένες πιθανότητες όλων των δυνατών τιμών της διακριτής μεταβλητής. Στη συνέχεια, μπορούμε να εφαρμόσουμε τη συνάρτηση softmax ώστε να εξάγουμε το διάνυσμα $\hat{\boldsymbol{y}}$ με τις κανονικοποιημένες τιμές των πιθανοτήτων.

Η εμπρόσθια μετάδοση ξεκινάει με τον καθορισμό της αρχικής κατάστασης $\boldsymbol{h}^{(0)}$. Έπειτα, για κάθε χρονικό βήμα $t = 1$ έως $t = \tau$ εφαρμόζουμε τις ακόλουθες εξισώσεις:

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)})$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)})$$

όπου οι παράμετροι είναι τα bias διανύσματα $\boldsymbol{b}$ και $\boldsymbol{c}$ μαζί με της μήτρες βαρών $\boldsymbol{U}, \boldsymbol{V}$ και $\boldsymbol{W}$ για τις διασυνδέσεις εισόδου – κρυφού στρώματος, κρυφού στρώματος – εξόδου και κρυφού στρώματος – κρυφού στρώματος αντίστοιχα.

Με τον καθορισμό της συνάρτησης απωλειών L, υπολογίζουμε τις μερικές παραγώγους ως προς τις παραμέτρους του μοντέλου. Ο υπολογισμός του gradient περιέχει ένα εμπρόσθιο πέρασμα, από τα αριστερά προς τα δεξιά στον χρονικά ανεπτυγμένο υπολογιστικό γράφο, και στη συνέχεια μια οπισθοδρόμηση (backward propagation), κινούμενοι από τα δεξιά προς τα αριστερά σε αυτόν.

Η πολυπλοκότητα χρόνου είναι $O(\tau)$ και δεν μπορεί να μειωθεί με παραλληλοποίηση λόγω της εγγενούς ακολουθιακής ανάπτυξης του γράφου – κάθε βήμα μπορεί να υπολογιστεί μόνο μετά το προηγούμενο. Οι καταστάσεις που υπολογίζονται στο εμπρόσθιο πέρασμα θα πρέπει να αποθηκεύονται μέχρι να χρησιμοποιηθούν κατά την οπισθοδρόμηση, και συνεπώς η πολυπλοκότητα μνήμης είναι και αυτή $O(\tau)$. Ο αλγόριθμος που χρησιμοποιείται για τον υπολογισμό των gradients στον ανεπτυγμένο γράφο ονομάζεται BPTT (back-propagation through time) και είναι υπολογιστικά ακριβός, με πολυπλοκότητα $O(\tau)$. [7, σσ 373–381]

6.2 Long Short-Term Memory Νευρωνικά Δίκτυα

Όπως είπαμε, ένα από τα βασικά χαρακτηριστικά των RNNs είναι η ικανότητά τους να συνδέουν προγενέστερη πληροφορία με τον τρέχοντα στόχο, όπως για παράδειγμα η χρησιμοποίηση προηγούμενων frame βίντεο για την κατανόηση του τρέχοντος frame. Μπορούν, όμως, τα RNNs να υλοποιήσουν αυτή τη λειτουργία;

Πολλές φορές, χρειαζόμαστε να κοιτάξουμε μόνο στις πρόσφατες πληροφορίες για να εκτιμήσουμε το παρόν. Για παράδειγμα, σε ένα γλωσσικό μοντέλο που θα του ζητούσαμε να προβλέψει την τελευταία λέξη στη φράση «τα σύννεφα είναι στον *ουρανό*», δεν χρειαζόμαστε περισσότερη πληροφορία – είναι εμφανές ότι η επόμενη λέξη θα είναι «ουρανός». Σε τέτοιες περιπτώσεις που το διάστημα είναι μικρό μεταξύ των σχετικών πληροφοριών και της θέσης της ζητούμενης εκτίμησης, τα RNNs μπορούν να μάθουν να χρησιμοποιούν την προγενέστερη πληροφορία.

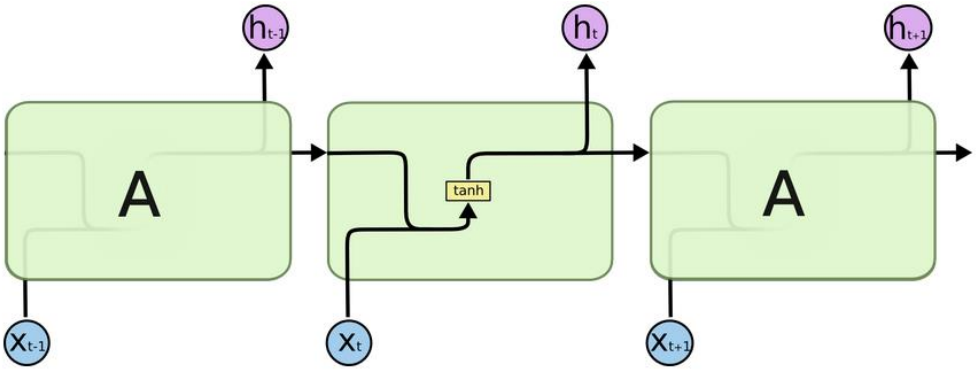
Υπάρχουν, βέβαια, περιπτώσεις που χρειαζόμαστε μεγαλύτερο συναφές περιεχόμενο για να βγάλουμε συμπέρασμα. Έστω ότι θέλαμε να προβλέψουμε την τελευταία λέξη σε ένα κείμενο που περιείχε ένα σύνολο προτάσεων «Μεγάλωσα στη Γαλλία… Μιλάω με άνεση *Γαλλικά*». Οι πρόσφατες πληροφορίες υποδηλώνουν ότι η επόμενη λέξη είναι πιθανότατα το όνομα μιας γλώσσας, αλλά αν θέλουμε να βρούμε τη γλώσσα αυτή, χρειαζόμαστε το ευρύτερο πλαίσιο (context) της «Γαλλίας» από προγενέστερες πληροφορίες. Το διάστημα μεταξύ της ζητούμενης εκτίμησης και των απαραίτητων για αυτήν πληροφοριών μπορεί να γίνει πολύ μεγάλο.

Δυστυχώς, όσο το διάστημα αυτό μεγαλώνει, τόσο πιο δύσκολη γίνεται για τα RNNs η ανάπτυξη των χρήσιμων συνδέσεων μεταξύ των πληροφοριών. Θεωρητικά, τα Recurrent Neural Networks είναι απολύτως ικανά να χειριστούν τέτοιες μεγάλων διαστημάτων εξαρτήσεις με κατάλληλη επιλογή των παραμέτρων τους. Στην πράξη, δυστυχώς, τα RNNs δεν φαίνονται ικανά να τις μάθουν [31].

Η απάντηση σε αυτές τις δυσκολίες έρχεται με τα Long Short Term Memory δίκτυα, ή απλά LSTMs, τα οποία έχουν την ικανότητα να μαθαίνουν εξαρτήσεις μεταξύ πληροφοριών που απέχουν μεγάλα χρονικά διαστήματα. Η διατύπωσή τους έγινε από τους Hochreiter και Schmidhuber το 1997 [32], και από τότε έχουν εξελιχθεί και χρησιμοποιηθεί από αρκετούς επιστήμονες και μηχανικούς. Το ιδιαίτερο χαρακτηριστικό τους είναι ότι λειτουργούν εξαιρετικά σε μια πληθώρα προβλημάτων, σε διαφορετικά πλαίσια.

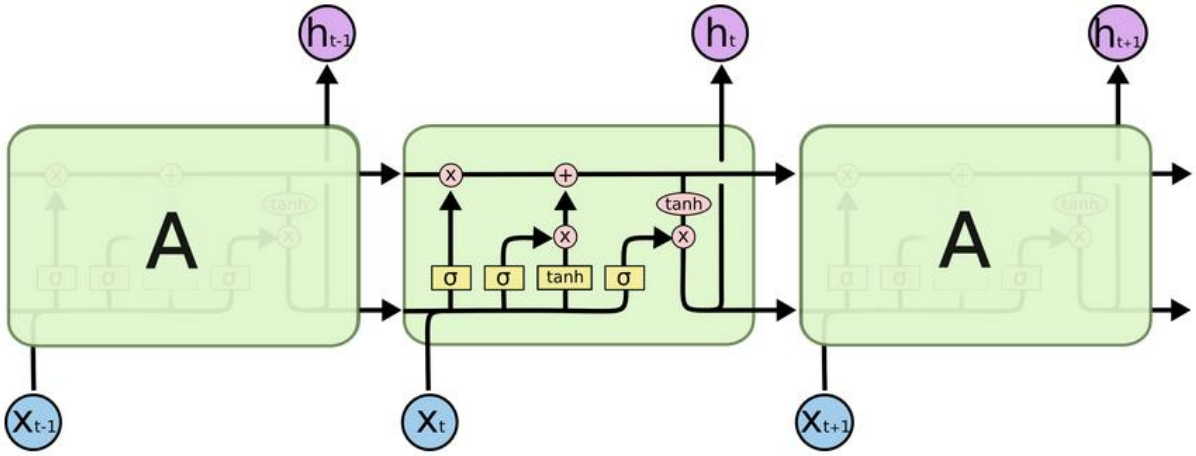
Τα LSTMs είναι σχεδιασμένα με τρόπο τέτοιο, ώστε να αντεπεξέρχονται στο πρόβλημα των εξαρτήσεων μακριών διαστημάτων. Μπορούμε να πούμε ότι, το να θυμούνται την πληροφορία για μακρές περιόδους, είναι η κανονική τους συμπεριφορά, όχι κάτι για το οποίο πρέπει να γίνει ιδιαίτερη προσπάθεια για να το καταφέρουν.

Ένα τυπικό RNN με απλή δομή, όπως ένα μοναδικό tanh στρώμα, φαίνεται ακολούθως:



Εικόνα 6.4: Ανεπτυγμένος γράφος τυπικού RNN με ένα tanh layer [33]

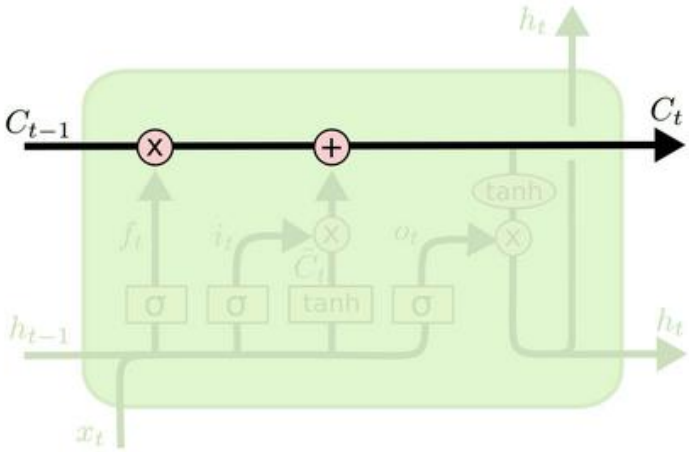
Τα LSTMs έχουν και αυτά αυτή την αρχιτεκτονική, αλλά το επαναλαμβανόμενο βασικό στοιχείο (module) έχει διαφορετική δομή. Στη θέση του ενός layer, το νευρωνικό δίκτυο έχει τέσσερα που αλληλεπιδρούν μεταξύ τους με έναν πολύ ιδιαίτερο τρόπο:



Εικόνα 6.5: Διαγραμματικό ανάπτυγμα του βασικού δομικού στοιχείου του LSTM

Στο παραπάνω διάγραμμα, κάθε γραμμή μεταφέρει ένα ολόκληρο διάνυσμα από την έξοδο ενός κόμβου στην είσοδο κάποιου άλλου. Οι ροζ κύκλοι αναπαριστούν πράξεις που γίνονται ανά σημείο, όπως η πρόσθεση διανυσμάτων, ενώ τα κίτρινα πλαίσια είναι τα στρώματα του νευρωνικού δικτύου, τις παραμέτρους των οποίων μαθαίνει κατά την εκπαίδευση. Οι γραμμές που ενώνονται συμβολίζουν τη συνένωση, ενώ εκείνες που διασπώνται δείχνουν ότι το περιεχόμενο αντιγράφεται και οδηγείται σε διαφορετικά σημεία.

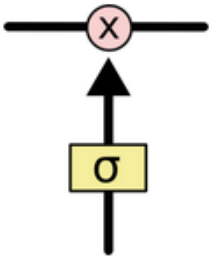
Το κλειδί των LSTMs είναι το κύτταρο κατάστασης (cell state), η οριζόντια γραμμή που διατρέχει το επάνω μέρος του διαγράμματος:



Εικόνα 6.6: Κύτταρο κατάστασης του LSTM

Το κύτταρο κατάστασης είναι σαν ένας μεταφορικός ιμάντας. Διατρέχει όλον τον ανεπτυγμένο γράφο με μόνο μικρές γραμμικές αλληλεπιδράσεις. Είναι εύκολο για την πληροφορία να ρεύσει μέσω αυτού ανεπηρέαστη.

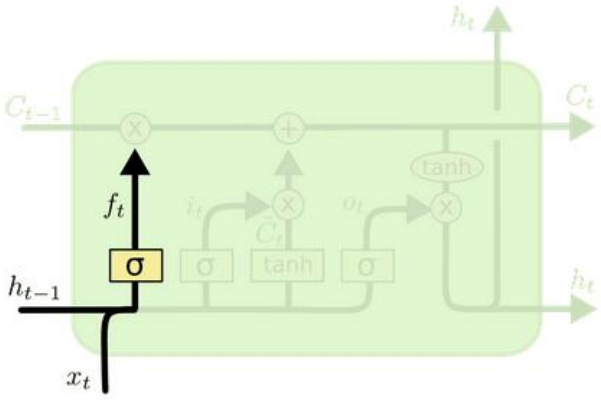
Το LSTM έχει την ικανότητα να αφαιρέσει ή να προσθέσει πληροφορία στο κύτταρο κατάστασης, μέσω των δομικών στοιχείων που ονομάζονται πύλες (gates). Οι πύλες ελέγχουν ποια πληροφορία θα περάσει και ποια όχι. Αποτελούνται από ένα σιγμοειδές στρώμα νευρώνων και υλοποιούν έναν ανά σημείο πολλαπλασιασμό:



Εικόνα 6.7: Τυπική πύλη του LSTM

Οι σιγμοειδείς νευρώνες δίνουν στην έξοδό τους μια τιμή από 0 έως 1, η οποία περιγράφει την ποσότητα από κάθε στοιχείο που θα πρέπει να αφεθεί να περάσει. Μια μηδενική τιμή σημαίνει “μην αφήσεις τίποτα να περάσει”, ενώ μια τιμή ίση με 1 “άσε οτιδήποτε να περάσει”. Ένα LSTM έχει τρεις τέτοιες πύλες που προστατεύουν και ελέγχουν το κύτταρο κατάστασης.

Το πρώτο βήμα για το LSTM είναι να αποφασίσουμε ποια πληροφορία είναι εκείνη που θα αφαιρέσουμε από το κύτταρο κατάστασης. Αυτή η απόφαση γίνεται μέσω ενός σιγμοειδούς στρώματος που ονομάζεται «στρώμα πύλης λήθης» (forget gate layer). Αυτή βλέπει το $\boldsymbol{h_{t-1}}$ και το $\boldsymbol{x_t}$ και δίνει έναν αριθμό από 0 έως 1 στην έξοδό της για κάθε έναν αριθμό του κυττάρου κατάστασης $\boldsymbol{C_{t-1}}$. Το 1 σημαίνει «κράτησέ το πλήρως» ενώ το 0 «ξέχασέ το εξολοκλήρου»:



Εικόνα 6.8: Στρώμα πύλης λήθης του LSTM

Η εξίσωση που αναπαριστά το στρώμα αυτό μπορεί να εκφραστεί ως εξής:

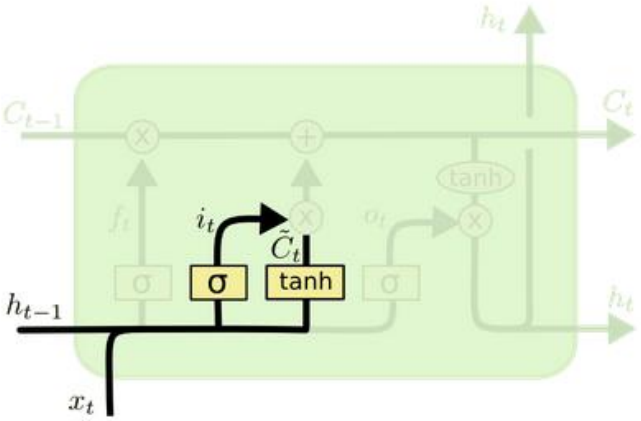
$$\boldsymbol{f}_t = \sigma(\boldsymbol{W}_f[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_f)$$

όπου $\boldsymbol{W_f}$ και $\boldsymbol{b_f}$ ο πίνακας των βαρών και το διάνυσμα των bias όρων του σιγμοειδούς στρώματος νευρώνων αντίστοιχα.

Επιστρέφοντας στο παράδειγμα της μοντελοποίησης γλώσσας στο οποίο προσπαθούμε να προβλέψουμε την επόμενη λέξη από τις προηγούμενες, το κύτταρο

κατάστασης θα μπορούσε να περιέχει το γένος του τρέχοντος υποκειμένου ώστε να χρησιμοποιηθούν οι σωστές αντωνυμίες. Όταν δούμε ένα νέο υποκείμενο, μπορούμε να ξεχάσουμε το γένος του παλιού.

Το επόμενο βήμα είναι να αποφασίσουμε ποια από τη νέα πληροφορία θα αποθηκεύσουμε στο κύτταρο κατάστασης. Η απόφαση αυτή αποτελείται από δύο μέρη. Πρώτα, ένα σιγμοειδές στρώμα που ονομάζουμε «στρώμα πύλης εισόδου» (input gate layer) αποφασίζει ποιες τιμές θα ανανεώσουμε και στη συνέχεια ένα $\tanh$ στρώμα δημιουργεί το διάνυσμα των νέων υποψήφιων τιμών $\widetilde{\boldsymbol{C}}_t$ που προορίζονται να προστεθούν στο κύτταρο κατάστασης:



Εικόνα 6.9: Στρώμα πύλης εισόδου και tanh στρώμα του LSTM

Οι εξισώσεις που περιγράφουν τα δύο στρώματα είναι οι ακόλουθες:

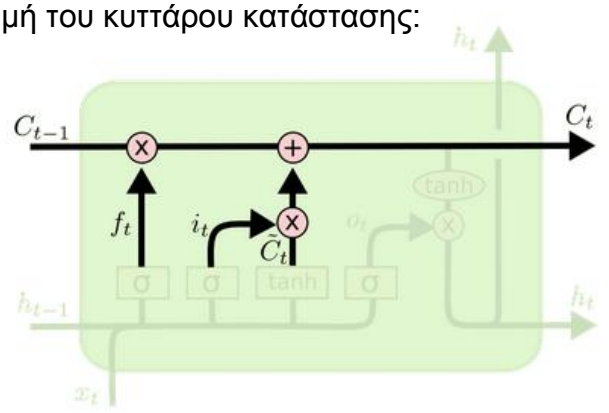
$$\boldsymbol{i}_t = \sigma(\boldsymbol{W}_i[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_i)$$

$$\widetilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{W}_C[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_C)$$

Στο παράδειγμά μας θα θέλαμε να προσθέσουμε το γένος του νέου υποκειμένου στο κύτταρο κατάστασης προς αντικατάσταση του προηγούμενου, το οποίο ξεχνάμε.

Είναι ώρα, λοιπόν, να ανανεώσουμε την κατάσταση του κυττάρου από την παλιά, $\boldsymbol{C}_{t-1}$, στην καινούρια, $\boldsymbol{C}_t$. Τα προηγούμενα βήματα έχουν ήδη αποφασίσει για το τι θα συμβεί και απομένει η υλοποίησή του.

Πολλαπλασιάζουμε την παλιά κατάσταση με $\boldsymbol{f}_t$, ξεχνώντας τα στοιχεία που αποφασίσαμε ότι θα ξεχάσουμε πρωτύτερα. Έπειτα προσθέτουμε το γινόμενο $\boldsymbol{i}_t\widetilde{\boldsymbol{C}}_t$, τις νέες υποψήφιες τιμές δηλαδή κανονικοποιημένες ανάλογα με το πόσο αποφασίσαμε να ανανεώσουμε κάθε τιμή του κυττάρου κατάστασης:



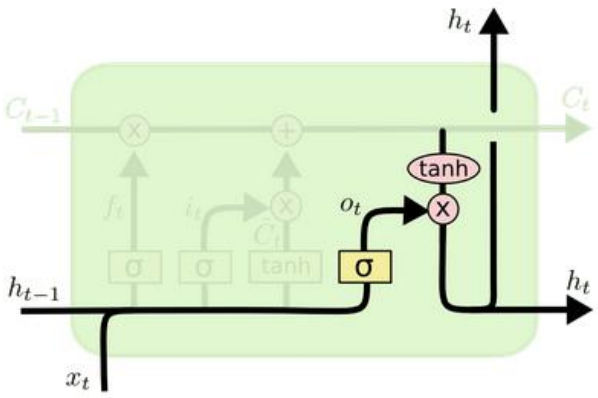
Εικόνα 6.10: Ανανέωση του κυττάρου κατάστασης του LSTM

Η εξίσωση που περιγράφει την ανανέωση του κυττάρου κατάστασης είναι η ακόλουθη:

$$\boldsymbol{C}_t = \boldsymbol{f}_t \boldsymbol{C}_{t-1} + \boldsymbol{i}_t \widetilde{\boldsymbol{C}}_t$$

Στην περίπτωση της μοντελοποίησης γλώσσας, εδώ είναι το σημείο στο οποίο αφήνουμε την πληροφορία που αφορά το γένος του παλιού υποκειμένου και στη θέση της προσθέτουμε νέα πληροφορία, όπως αποφασίσαμε στα προηγούμενα βήματα.

Τέλος, θα πρέπει να αποφασίσουμε ποια θα είναι η έξοδος $\boldsymbol{h}_t$. Η έξοδός μας θα βασίζεται στο κύτταρο κατάστασης, αλλά θα είναι μια φιλτραρισμένη εκδοχή του. Πρώτα, χρησιμοποιούμε ένα σιγμοειδές στρώμα το οποίο αποφασίζει ποια μέρη του κυττάρου κατάστασης θα προωθήσουμε στην έξοδο. Στη συνέχεια, εφαρμόζουμε στις τιμές του κυττάρου κατάστασης τη συνάρτηση $\tanh$, ώστε να κινηθούν στο διάστημα από -1 έως 1, και πολλαπλασιάζουμε με την έξοδο του σιγμοειδούς στρώματος για να δώσουμε στην έξοδο μόνο τα μέρη που αποφασίσαμε:



Εικόνα 6.11: Διαμόρφωση της εξόδου του LSTM

Οι εξισώσεις που αναπαριστούν τα παραπάνω είναι οι ακόλουθες:

$$\boldsymbol{o}_t = \sigma(\boldsymbol{W}_o[\boldsymbol{h}_{t-1}, \boldsymbol{x}_t] + \boldsymbol{b}_o)$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \tanh(\boldsymbol{C}_t)$$

Για το παράδειγμα μοντελοποίησης γλώσσας, από τη στιγμή που μόλις είδαμε ένα υποκείμενο, πιθανόν να θέλουμε να εξάγουμε πληροφορίες για το σχετικό ρήμα. Για παράδειγμα, το μοντέλο μας θα μπορούσε να εξάγει την πληροφορία αν το υποκείμενο είναι στον ενικό ή τον πληθυντικό, ώστε να γνωρίζουμε τη σωστή κλίση του ρήματος, στην περίπτωση που αυτό θα ακολουθήσει. [33]

Παραπάνω, αναπτύχθηκε βήμα προς βήμα η λειτουργία ενός τυπικού LSTM. Από τη διατύπωσή τους, το 1997, μέχρι σήμερα έχουν δημιουργηθεί πολλές διαφορετικές παραλλαγές των LSTMs. Τα δίκτυα αυτά έχουν εκπληκτικές αποδόσεις σε πάρα πολλά προβλήματα. Αυτό οφείλεται στην ικανότητα που κληρονομούν ως RNNs να εξάγουν διασυνδέσεις από τα δεδομένα που κινούνται στον χρόνο, και στη δυνατότητά τους να διατηρούν τις διασυνδέσεις αυτές ακόμα και αν το χρονικό διάστημα που μεσολαβεί είναι αρκετά μεγάλο. Στην επόμενη παράγραφο μελετάμε τη χρήση ενός LSTM για τη μοντελοποίηση γλώσσας και εξετάζουμε κάποια πειραματικά αποτελέσματα αυτού.

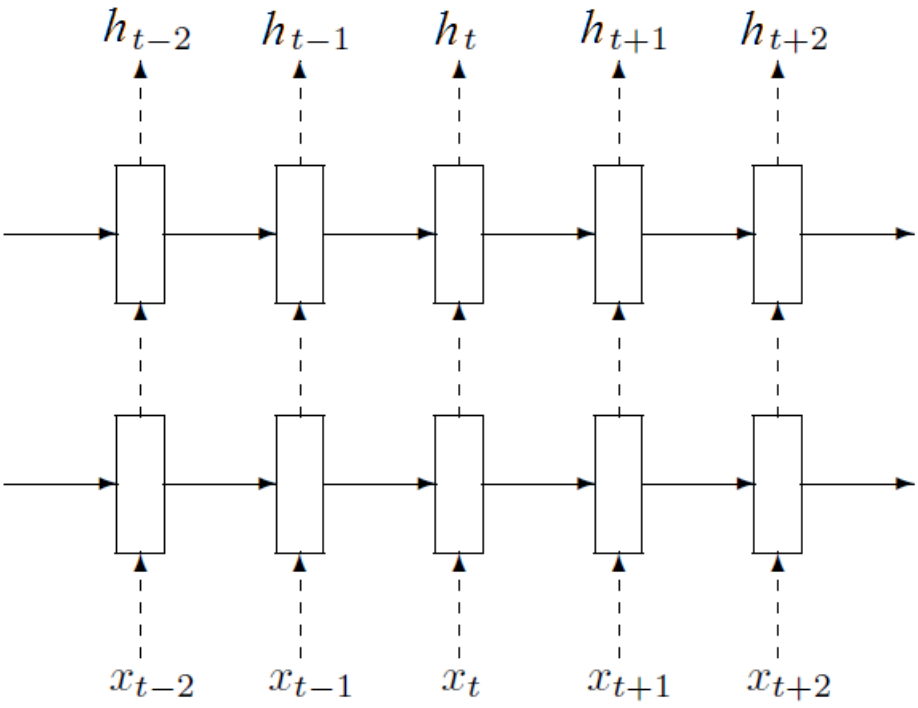
6.3 Μελέτη υλοποίησης LSTM με Dropout για τη μοντελοποίηση γλώσσας

Στην παράγραφο αυτή θα μελετήσουμε την ανάπτυξη ενός LSTM για τη μοντελοποίηση γλώσσας. Θα χρησιμοποιήσουμε το μοντέλο που έχει αναπτυχθεί από την ομάδα "TensorFlow" της Google και θα διερευνήσουμε πειραματικά τη συμβολή του Dropout σε αυτό. Ο πλήρης κώδικας είναι προσβάσιμος εδώ, [34].

Ας δούμε, αρχικά, το πρόβλημα που θέλουμε να αντιμετωπίσουμε με το LSTM μοντέλο μας. Ο στόχος είναι να αναπτύξουμε ένα πιθανοτικό μοντέλο που εκχωρεί πιθανότητες σε προτάσεις. Αυτό το επιτυγχάνει προβλέποντας τις επόμενες λέξεις ενός κειμένου που του δίνεται από τις προηγούμενες. Για τον σκοπό αυτό, θα χρησιμοποιήσουμε το Penn Tree Bank (ή PTB) dataset [35], ένα δημοφιλές μέτρο για την απόδοση αυτών των μοντέλων. Να αναφέρουμε εδώ ότι η μοντελοποίηση γλώσσας είναι το κλειδί για πολλά ενδιαφέροντα προβλήματα όπως η αναγνώριση ομιλίας ή η αυτόματη μετάφραση κειμένου με τον υπολογιστή.

Η δημοσίευση την οποία ακολουθεί ο σχεδιασμός του μοντέλου είναι αυτή των Wojciech Zaremba, Ilya Sutskever, Oriol Vinyals, τον Σεπτέμβρη του 2014 [36], στην οποία επιτυγχάνεται πολύ καλή απόδοση στο PTB dataset. Η κύρια συμβολή αυτής της δημοσίευσης είναι η περιγραφή του τρόπου με τον οποίο η εφαρμογή του dropout στα LSTMs μπορεί να επιτύχει τη μείωση του overfitting.

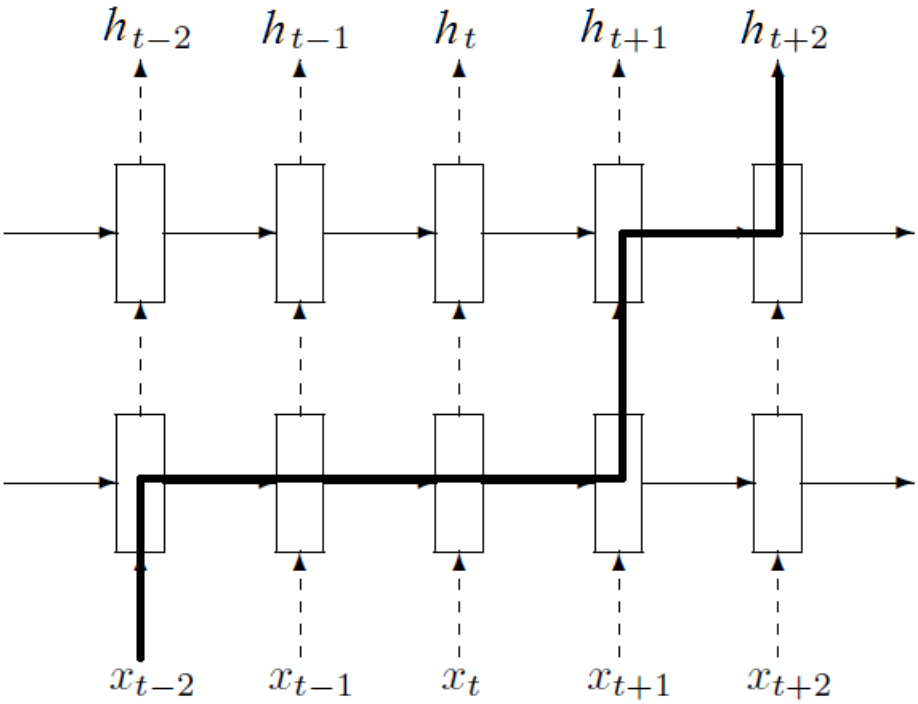
Η κύρια ιδέα είναι να εφαρμόσουμε dropout μόνο στις μη επαναλαμβανόμενες συνδέσεις ενός RNN. Στην παρακάτω εικόνα, ακολουθώντας τα χρονικά βήματα από $t-2$ έως $t+2$, συμβολίζονται με διακεκομμένη γραμμή οι συνδέσεις στις οποίες χρησιμοποιείται dropout και με συνεχή γραμμή εκείνες στις οποίες δεν εφαρμόζουμε εξομάλυνση:



Εικόνα 6.12: Χρήση του Regularization σε ένα RNN

Ουσιαστικά, με το dropout απομειώνουμε την πληροφορία που μεταφέρουν οι νευρώνες ωθώντας τους έτσι στο να οικοδομήσουν νοηματικά ισχυρότερες συνδέσεις μεταξύ τους. Παράλληλα, δεν επιθυμούμε να διαγράψουμε πληροφορία που προέρχεται από τα προηγούμενα βήματα καθώς είναι ιδιαιτέρως σημαντική η ανάμνηση δεδομένων που

εμφανίστηκαν ακόμα και αρκετά βήματα πιο πριν. Στην παρακάτω εικόνα, παρατηρούμε τη διαδρομή της πληροφορίας που εμφανίστηκε τη χρονική στιγμή $t-2$ και χρησιμοποιήθηκε στην πρόβλεψη στο βήμα $t+2$:



Εικόνα 6.13: Διαδρομή της πληροφορίας σε ένα LSTM

Μπορούμε να συμπεράνουμε ότι η πληροφορία μας απομειώθηκε, μέσω του dropout, ακριβώς $L+1$ φορές, όπου L είναι το πλήθος των στρωμάτων του δικτύου. Ακόμα, ο αριθμός των απομειώσεων είναι ανεξάρτητος των χρονικών βημάτων που παρεμβλήθηκαν μεταξύ των δύο στιγμών.

Το τυπικό dropout θα διατάραζε τις recurrent διασυνδέσεις και θα έκανε δύσκολη για το LSTM την εκμάθηση του τρόπου με τον οποίο πρέπει να αποθηκεύσει πληροφορία από το παρελθόν. Μη χρησιμοποιώντας εξομάλυνση στις επαναλαμβανόμενες συνδέσεις, το LSTM επωφελείται από τη μείωση του overfitting, χωρίς παράλληλα να θυσιάζει πολύτιμη ικανότητα απομνημόνευσης.

Το PTB dataset αριθμεί 929 χιλιάδες λέξεις στο σύνολο εκπαίδευσης, 73 χιλιάδες στο σύνολο επαλήθευσης και 82 χιλιάδες στο σύνολο ελέγχου. Υπάρχουν 10 χιλιάδες διαφορετικές λέξεις στο λεξιλόγιό του. Μελετήσαμε την εκπαίδευση ενός LSTM σε αυτό. Το κριτήριο που χρησιμοποιούμε για τη μέτρηση της απόδοσης του γλωσσικού μοντέλου μας είναι η perplexity (σύγχυση) την οποία συχνά συμβολίζουμε με PP. Η perplexity ενός μοντέλου στο σύνολο ελέγχου είναι η αντίστροφη πιθανότητα που δίνεται στο σύνολο αυτό, κανονικοποιημένη με το πλήθος των λέξεών του. Για ένα σύνολο ελέγχου $W = w_1 w_2 \dots w_N$ θα είναι:

$$PP(W) = P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} = \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}$$

και χρησιμοποιώντας τον κανόνα της αλυσίδας:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \dots w_{i-1})}}$$

Βλέπουμε ότι όσο μεγαλύτερη είναι η δεσμευμένη πιθανότητα που δίνεται στην ακολουθία των λέξεων, τόσο μικρότερο είναι το μέτρο της perplexity. Έτσι, αναμένουμε από ένα αποδοτικό γλωσσικό μοντέλο να έχει όσο το δυνατόν μικρότερη τιμή της perplexity στο σύνολο ελέγχου [37, σ 42].

Επιστρέφοντας στο μοντέλο μας, θα χρησιμοποιήσουμε ένα LSTM με 2 στρώματα και ανεπτυγμένο σε 35 χρονικά βήματα. Αρχικοποιούμε με 0 τις καταστάσεις των κρυφών στρωμάτων. Χρησιμοποιούμε κάθε φορά τις τελικές κρυφές καταστάσεις του τρέχοντος minibatch ως αρχικές κρυφές καταστάσεις του επόμενου minibatch, στη χρονική ακολουθία που αποτελείται από 35 βήματα. Το μέγεθος κάθε minibatch είναι 20 λέξεις.

Το LSTM που χρησιμοποιήσαμε έχει 650 νευρώνες σε κάθε στρώμα και, όπως έχουμε δει στην προηγούμενη παράγραφο, κάθε στρώμα αποτελείται από 4 στρώματα που αλληλεπιδρούν με έναν ιδιαίτερο τρόπο μεταξύ τους, 3 σιγμοειδή και ένα $\tanh$ layer.

Ας δούμε, λοιπόν, τα βασικά σημεία της υλοποίησης του LSTM μοντέλου στο TensorFlow. Ξεκινάμε με την υλοποίηση της κλάσης που αναφέρεται στο μοντέλο μας:

```
class PTBModel(object):
  """The PTB model."""

  def __init__(self, is_training, config, input_):
    self._is_training = is_training
    self._input = input_
    self._rnn_params = None
    self._cell = None
    self.batch_size = input_.batch_size
    self.num_steps = input_.num_steps
    size = config.hidden_size
    vocab_size = config.vocab_size

    with tf.device("/cpu:0"):
      embedding = tf.get_variable(
          "embedding", [vocab_size, size], dtype=data_type())
      inputs = tf.nn.embedding_lookup(embedding, input_.input_data)

    if is_training and config.keep_prob < 1:
      inputs = tf.nn.dropout(inputs, config.keep_prob)

    output, state = self._build_rnn_graph_lstm(inputs, config, is_training)

    softmax_w = tf.get_variable(
        "softmax_w", [size, vocab_size], dtype=data_type())
    softmax_b = tf.get_variable("softmax_b", [vocab_size], dtype=data_type())
    logits = tf.nn.xw_plus_b(output, softmax_w, softmax_b)
```

```
    # Reshape logits to be a 3-D tensor for sequence loss
    logits = tf.reshape(logits, [self.batch_size, self.num_steps, vocab_size])

    # Use the contrib sequence loss and average over the batches
    loss = tf.contrib.seq2seq.sequence_loss(
        logits,
        input_.targets,
        tf.ones([self.batch_size, self.num_steps], dtype=data_type()),
        average_across_timesteps=False,
        average_across_batch=True)

    # Update the cost
    self._cost = tf.reduce_sum(loss)
    self._final_state = state

    if not is_training:
      return

    self._lr = tf.Variable(0.0, trainable=False)
    tvars = tf.trainable_variables()
    grads, _ = tf.clip_by_global_norm(tf.gradients(self._cost, tvars),
                                      config.max_grad_norm)
    optimizer = tf.train.GradientDescentOptimizer(self._lr)
    self._train_op = optimizer.apply_gradients(
        zip(grads, tvars),
        global_step=tf.train.get_or_create_global_step())

    self._new_lr = tf.placeholder(
        tf.float32, shape=[], name="new_learning_rate")
    self._lr_update = tf.assign(self._lr, self._new_lr)
```

Το αντικείμενο που θα φτιαχτεί, και θα ανήκει στην κλάση αυτή, θα λάβει ως ορίσματα μια boolean “is_training”, ένα αντικείμενο config με όλες τις ρυθμίσεις που έχουμε επιλέξει και ασφαλώς ένα αντικείμενο που περιέχει την είσοδο. Στη συνέχεια γίνεται έλεγχος για τη ρύθμιση σχετικά με το dropout και εφαρμόζεται ή όχι αυτό, στα δεδομένα της εισόδου. Έπειτα, το κύτταρο κατάστασης καθώς και η έξοδος υπολογίζονται μέσω της συνάρτησης _build_rnn_graph_lstm():

```
  def _build_rnn_graph_lstm(self, inputs, config, is_training):
    """Build the inference graph using canonical LSTM cells."""

    def make_cell():
      cell = tf.contrib.rnn.BasicLSTMCell(
          config.hidden_size, forget_bias=0.0, state_is_tuple=True,
          reuse=not is_training)
      if is_training and config.keep_prob < 1:
        cell = tf.contrib.rnn.DropoutWrapper(
            cell, output_keep_prob=config.keep_prob)
      return cell
```

```
    cell = tf.contrib.rnn.MultiRNNCell(
        [make_cell() for _ in range(config.num_layers)], state_is_tuple=True)

    self._initial_state = cell.zero_state(config.batch_size, data_type())
    state = self._initial_state

    outputs = []
    with tf.variable_scope("RNN"):
      for time_step in range(self.num_steps):
        if time_step > 0: tf.get_variable_scope().reuse_variables()
        (cell_output, state) = cell(inputs[:, time_step, :], state)
        outputs.append(cell_output)
        output = tf.reshape(tf.concat(outputs, 1), [-1, config.hidden_size])

    return output, state
```

Βλέπουμε ότι και εδώ λαμβάνουμε υπόψη τη ρύθμιση για το dropout ώστε να το εφαρμόσουμε στα δεδομένα που θα προωθηθούν στην έξοδο. Επίσης, χρησιμοποιούνται οι μέθοδοι tf.contrib.rnn.BasicLSTMCell() όπως και tf.contrib.rnn.MultiRNNCell() από τη βιβλιοθήκη του TensorFlow για τα RNNs [38] για να διαμορφωθεί το μοντέλο και να επιστραφούν οι νέες τιμές για το κύτταρο κατάστασης και την έξοδο.

Επιστρέφοντας στον κώδικα της κλάσης του μοντέλου, παρατηρούμε ότι έχοντας δεδομένα τα output και state, υπολογίζονται οι κανονικοποιημένες με τη softmax τιμές (logits) και στη συνέχεια οι απώλειες (loss) του μοντέλου σε σχέση με τις τιμές-στόχους.

Στη συνέχεια, εφόσον πρόκειται για το σύνολο εκπαίδευσης, το κόστος που προκύπτει από τις απώλειες οδηγεί στον υπολογισμό των gradients και στην ανανέωση των τιμών των παραμέτρων με τον αλγόριθμο gradient descent.

Θα πρέπει να ορίσουμε την ακόλουθη μέθοδο, που λαμβάνει όλα τα δεδομένα του μοντέλου σε κάθε εποχή, εκχωρεί τα κατάλληλα Tensors και καλεί την εκτέλεση του Session για τον υπολογισμό του μέτρου της perplexity σε κάθε εποχή:

```
def run_epoch(session, model, eval_op=None, verbose=False):
    """Runs the model on the given data."""
    costs = 0.0
    iters = 0
    state = session.run(model.initial_state)

    fetches = {
        "cost": model.cost,
        "final_state": model.final_state,
    }
    if eval_op is not None:
        fetches["eval_op"] = eval_op


    for step in range(model.input.epoch_size):
        feed_dict = {}
```

```
        for i, (c, h) in enumerate(model.initial_state):
            feed_dict[c] = state[i].c
            feed_dict[h] = state[i].h

        vals = session.run(fetches, feed_dict)
        cost = vals["cost"]
        state = vals["final_state"]

        costs += cost
        iters += model.input.num_steps

    return np.exp(costs / iters)
```

Τέλος, στο κύριο μέρος του κώδικα αναμένουμε να υπάρχει η ακόλουθη κύρια λειτουργία:

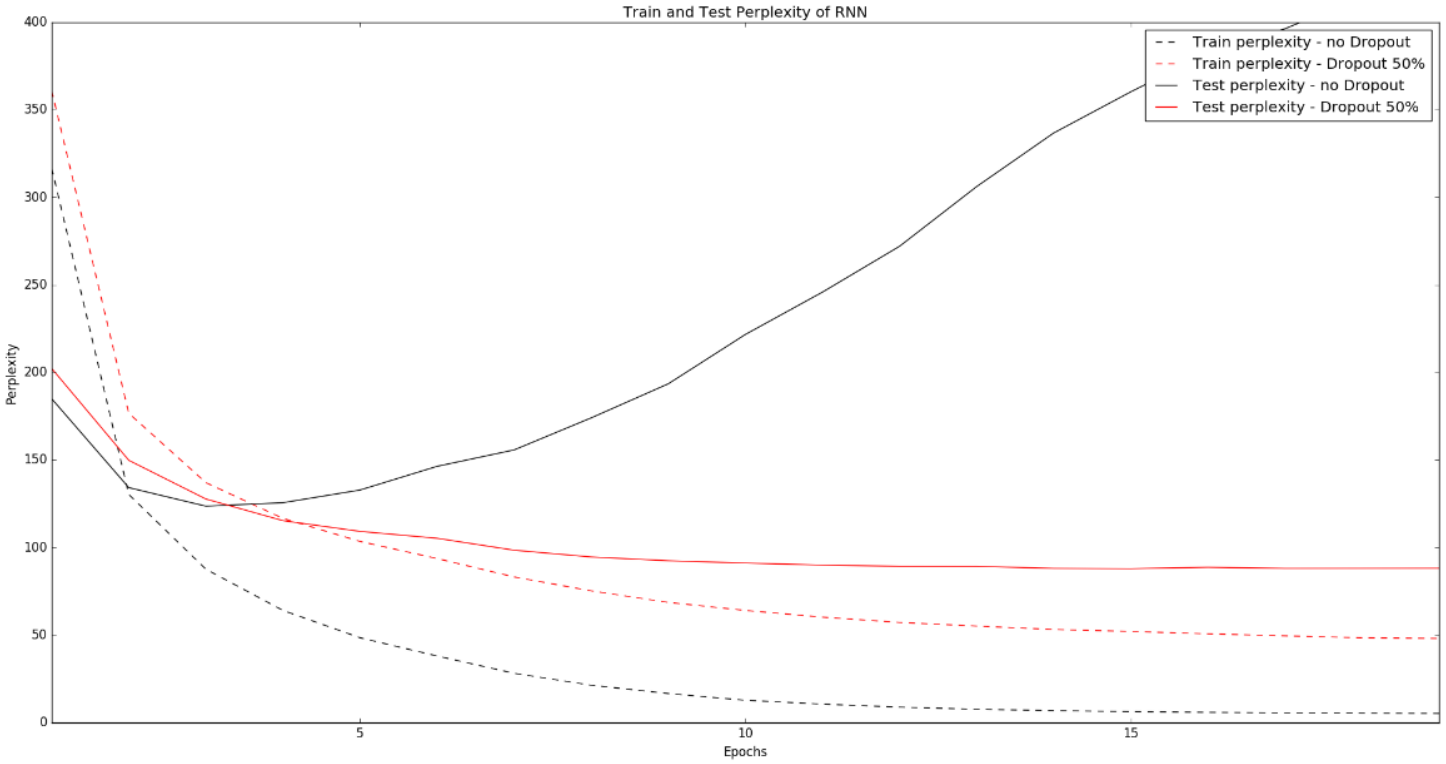
```
with tf.Session() as session:

    for i in range(config.max_max_epoch):
        lr_decay = config.lr_decay ** max(i + 1 - config.max_epoch, 0.0)
        m.assign_lr(session, config.learning_rate * lr_decay)

        train_perplexity = run_epoch(session, m, eval_op=m.train_op,
                                     verbose=True)

    test_perplexity = run_epoch(session, mtest)
    print("Test Perplexity: %.3f" % test_perplexity)
```

Εδώ βλέπουμε ότι χρησιμοποιείται μια μέθοδος για την απομείωση της τιμής του learning rate σε κάθε εποχή με σκοπό την αποτελεσματικότερη εκπαίδευση. Σε κάθε εποχή υπολογίζεται το μέτρο της perplexity για τα σύνολα εκπαίδευσης και ελέγχου. Το σύνολο του κώδικα που περιέχει πέρα από τα βασικά στοιχεία που αναλύθηκαν εδώ, όλα τα στοιχεία του μοντέλου καθώς και τον τρόπο με τον οποίο διαβάζονται τα δεδομένα του PTB dataset είναι διαθέσιμο εδώ, [34].

Χρησιμοποιήσαμε το παραπάνω μοντέλο για να εξετάσουμε τη σημασία που έχει το Dropout σε αυτό. Εκπαιδεύσαμε το LSTM για 20 εποχές, ξεκινήσαμε με learning rate ίσο με 1 και από την $6^{η}$ εποχή και μετά το μειώναμε με ένα συντελεστή 0.8 κάθε φορά.

Για να έχουμε τη δυνατότητα της σύγκρισης, εκπαιδεύσαμε δύο φορές το μοντέλο μας. Την πρώτη χωρίς τη χρήση regularization και τη δεύτερη ορίζοντας dropout 50% για τα δεδομένα στην είσοδο και στην έξοδο, όπως έχει αναπτυχθεί παραπάνω.

Μετρήσαμε την τιμή της perplexity σε κάθε εποχή τόσο για το σύνολο εκπαίδευσης όσο και για το σύνολο ελέγχου, και για τις δύο περιπτώσεις. Τα αποτελέσματα φαίνονται στο σχήμα που ακολουθεί. Με διακεκομμένη γραμμή είναι οι τιμές για το training set ενώ με συνεχή εκείνες για το test set:



Σχήμα 6.1: Perplexity του LSTM γλωσσικού μοντέλου με και χωρίς τη χρήση Dropout

Με κόκκινο χρώμα βλέπουμε τις καμπύλες που έχουν προκύψει με τη χρήση dropout ενώ με μαύρο εκείνες στις οποίες δεν έχει εφαρμοστεί κάποια εξομάλυνση. Είναι εμφανές το φαινόμενο του overfitting του μοντέλου όταν δεν χρησιμοποιείται regularization. Παρατηρούμε στις δύο μαύρες καμπύλες, ότι ενώ η perplexity του training set μειώνεται, εκείνη του test set αφού μειωθεί για μικρό αριθμό εποχών, αρχίζει να αυξάνεται με όλο και εντονότερη κλίση.

Πρόκειται για μια κλασική περίπτωση overtraining/overfitting του μοντέλου. Κάτι που δεν βλέπουμε σε καμία περίπτωση να συμβαίνει όταν εφαρμόζουμε regularization με dropout σε αυτό. Ακολουθώντας τις κόκκινες καμπύλες, παρατηρούμε ότι το μέτρο της perplexity στο σύνολο εκπαίδευσης μειώνεται με πιο αρμονικό τρόπο. Παράλληλα, η perplexity στο test set συνεχίζει την πτωτική της πορεία. Με άλλα λόγια, με την εφαρμογή του dropout το LSTM γλωσσικό μοντέλο συνέχιζε να αυξάνει την απόδοσή του στο σύνολο ελέγχου σε κάθε εποχή, βελτιώνοντας έτσι την ικανότητα του να γενικεύσει σε άγνωστα για αυτό δεδομένα.

7. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην εργασία αυτή ερευνήσαμε τα νευρωνικά δίκτυα από τα πιο κλασικά μέχρι εκείνα που διατυπώθηκαν πριν κάποια χρόνια και συνεχίζουν να εξελίσσουν την αρχιτεκτονική τους. Πέρα από την πλήρη παρουσίαση της θεωρίας και των βασικών εννοιών που διέπουν τους αλγορίθμους αυτούς, είδαμε τον τρόπο με τον οποίο μπορούμε να τους υλοποιήσουμε προγραμματιστικά.

Έτσι, διαπιστώσαμε με τη σειρά μας τη δυναμική των νευρωνικών δικτύων στο πεδίο της βαθιάς μηχανικής μάθησης. Ξεκινώντας με τα πλήρως συνδεδεμένα feedforward νευρωνικά δίκτυα, συνεχίζοντας με τα Convolutional Neural Networks, κλείνοντας με τα RNNs και τα LSTMs διαπιστώσαμε τα διαφορετικά χαρακτηριστικά των αλγορίθμων αυτών και την ικανότητά τους στην αντιμετώπιση ιδιαίτερων προβλημάτων.

Πιο συγκεκριμένα, διαμορφώσαμε διαφορετικές αρχιτεκτονικές βαθιών feedforward δικτύων για την αναγνώριση ψηφίων γραμμένων με το χέρι και εξετάσαμε τις διαφορές στην απόδοσή τους. Στη συνέχεια, προχωρήσαμε σε μια νέα αντιμετώπιση για το ίδιο πρόβλημα χρησιμοποιώντας ένα CNN για τον σκοπό αυτό. Όπως ήταν αναμενόμενο, καθώς βρισκόμασταν στο ευρύτερο πεδίο της αναγνώρισης εικόνας, το Convolutional Neural Network πέτυχε σημαντικά καλύτερη απόδοση στο πρόβλημα. Τέλος, για την παρουσίαση των δικτύων με, ίσως, την πιο δυναμική αρχιτεκτονική, των Recurrent Neural Networks, μελετήσαμε ένα πρόβλημα μοντελοποίησης γλώσσας και παρουσιάσαμε τον τρόπο με τον οποίο αναπτύσσουμε ένα Long Short-Term Memory δίκτυο για την αντιμετώπισή του.

Πέρα από την ανάπτυξη των αλγορίθμων βαθιάς μηχανικής μάθησης, ερευνήσαμε σε όλους τη σημασία της χρήσης regularization και συγκεκριμένα της μεθόδου Dropout. Καταλήξαμε στο συμπέρασμα ότι σε κάθε περίπτωση, από όλες τις αρχιτεκτονικές των feedforward δικτύων, μέχρι το Convolutional Neural Network και το LSTM, η απόδοση των μοντέλων βελτιώθηκε αισθητά με τη χρήση του Dropout.

Παρατηρήσαμε σε αρκετά διαφορετικά πολυστρωματικά δίκτυα ότι είχαμε σημαντικές μεταβολές στην απόδοση, όχι τόσο σε ένα δίκτυο σε σχέση με κάποιο άλλο, όσο στο ίδιο δίκτυο με την εφαρμογή ή όχι Dropout και ανάλογα τα στρώματα στα οποία το υλοποιούσαμε. Διαπιστώσαμε ότι η καλύτερη τακτική ήταν η εφαρμογή του τόσο στο στρώμα εισόδου όσο φυσικά και στα κρυφά στρώματα.

Την ίδια συμπεριφορά είχαμε στο Convolutional Neural Network που υλοποιήσαμε, όπου το Dropout στα πλήρως διασυνδεδεμένα στρώματα οδήγησε σε αισθητή αύξηση στην απόδοση του μοντέλου.

Ακόμα, στο LSTM που αναπτύχθηκε για τη μοντελοποίηση γλώσσας, παρατηρήσαμε την ανάγκη ύπαρξης εξομάλυνσης για την αποτροπή του overfitting στο μοντέλο. Χωρίς καμία τεχνική regularization στα μη επαναλαμβανόμενα στρώματα το σφάλμα του μοντέλου στο σύνολο ελέγχου άρχισε, από τις πρώτες κιόλας εποχές, να αυξάνεται διαρκώς όσο το σφάλμα του συνόλου εκπαίδευσης διαρκώς μειωνόταν. Γεγονός που δεν συνέβη όταν εφαρμόσαμε Dropout, όπου παρατηρήσαμε ότι το σφάλμα στο σύνολο ελέγχου μειωνόταν με το πέρασμα των εποχών, επισημαίνοντας τη βελτίωση στη γενικευτική ικανότητα του μοντέλου.

Κλείνοντας, λοιπόν, αναπτύξαμε μια πληθώρα αλγορίθμων βαθιάς μηχανικής μάθησης, νευρωνικά δίκτυα με βαθιές αρχιτεκτονικές ικανά για την αντιμετώπιση ενός μεγάλου εύρους προβλημάτων. Ερευνήσαμε και διαπιστώσαμε την ιδιαίτερη σημασία που έχει η εξομάλυνση στους αλγορίθμους αυτούς για την αποτροπή της υπερ-εκπαίδευσης, της υπερβολικής προσαρμογής τους, με άλλα λόγια, στα δεδομένα που χρησιμοποιούμε κατά τη μάθηση.

ΑΝΑΦΟΡΕΣ

[1] 'NP-completeness', Wikipedia. 29-Ιανουαρίου-2018.
[2] C. Daskalakis, P. W. Goldberg, και C. H. Papadimitriou, 'The Complexity of Computing a Nash Equilibrium', στο Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing, New York, NY, USA, 2006, σσ 71–78.
[3] S. Theodoridis, Machine Learning: A Bayesian and Optimization Perspective, 1 edition. Amsterdam: Elsevier, 2015.
[4] W. S. McCulloch και W. Pitts, 'A logical calculus of the ideas immanent in nervous activity', Bulletin of Mathematical Biophysics, τ. 5, τχ. 4, σσ 115–133, Δεκεμβρίου 1943.
[5] F. Rosenblatt, 'The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain', Psychological Review, σσ 65–386, 1958.
[6] F. Rosenblatt, Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms. Spartan Books, 1962.
[7] I. Goodfellow, Y. Bengio, και A. Courville, Deep Learning. Cambridge, Massachusetts: The MIT Press, 2016.
[8] K. Foote, 'A Brief History of Deep Learning', http://www.dataversity.net/brief-history-deep-learning/, 07-Φεβρουαρίου-2017.
[9] L. Deng και D. Yu, 'Deep Learning: Methods and Applications', SIG, τ. 7, τχ. 3–4, σσ 197–387, Ιουνίου 2014.
[10] T. Serre, G. Kreiman, M. Kouh, C. Cadieu, U. Knoblich, και T. Poggio, 'A quantitative theory of immediate visual recognition', Prog. Brain Res., τ. 165, σσ 33–56, 2007.
[11] M. A. Nielsen, 'Neural Networks and Deep Learning', http://neuralnetworksanddeeplearning.com, 2015.
[12] F. Dyson, 'A meeting with Enrico Fermi', Nature, τ. 427, τχ. 6972, σσ 297–297, Ιανουαρίου 2004.
[13] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, και R. Salakhutdinov, 'Dropout: A Simple Way to Prevent Neural Networks from Overfitting', Journal of Machine Learning Research, τ. 15, σσ 1929–1958, 2014.
[14] S. A. Scarpinelli Danijar Hafner, Erik Erwitt, Ariel, TensorFlow for Machine Intelligence. .
[15] 'TensorFlow', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/. [Ημερομηνία πρόσβασης: 24-Δεκεμβρίου-2017].
[16] 'Comparison of deep learning software', Wikipedia. 24-Δεκεμβρίου-2017.
[17] 'API Documentation', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/api_docs/. [Ημερομηνία πρόσβασης: 25-Δεκεμβρίου-2017].
[18] 'tf.Session', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/api_docs/python/tf/Session. [Ημερομηνία πρόσβασης: 25-Δεκεμβρίου-2017].
[19] 'Constants, Sequences, and Random Values', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/versions/r0.12/api_docs/python/constant_op/. [Ημερομηνία πρόσβασης: 26-Δεκεμβρίου-2017].
[20] 'tf.train.Saver', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/versions/r1.0/api_docs/python/tf/train/Saver. [Ημερομηνία πρόσβασης: 26-Δεκεμβρίου-2017].
[21] 'MNIST database', Wikipedia. 25-Δεκεμβρίου-2017.
[22] 'MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://yann.lecun.com/exdb/mnist/. [Ημερομηνία πρόσβασης: 31-Δεκεμβρίου-2017].
[23] 'MNIST For ML Beginners', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/get_started/mnist/beginners. [Ημερομηνία πρόσβασης: 31-Δεκεμβρίου-2017].
[24] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, και R. R. Salakhutdinov, 'Improving neural networks by preventing co-adaptation of feature detectors', arXiv preprint, τ. arXiv, Ιουλίου 2012.
[25] 'Module: tf.nn', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/api_docs/python/tf/nn. [Ημερομηνία πρόσβασης: 01-Ιανουαρίου-2018].
[26] 'tf.nn.dropout', TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/api_docs/python/tf/nn/dropout. [Ημερομηνία πρόσβασης: 01-Ιανουαρίου-2018].
[27] Y. Lecun, L. Bottou, Y. Bengio, και P. Haffner, 'Gradient-based learning applied to document recognition', Proceedings of the IEEE, τ. 86, τχ. 11, σσ 2278–2324, Νοεμβρίου 1998.
[28] 'What is visual recognition? | Clarifai'. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.clarifai.com/technology. [Ημερομηνία πρόσβασης: 07-Ιανουαρίου-2018].

[29] ‘CS231n Convolutional Neural Networks for Visual Recognition’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://cs231n.github.io/convolutional-networks/. [Ημερομηνία πρόσβασης: 11-Ιανουαρίου-2018].
[30] Ujjwal Karn, ‘An Intuitive Explanation of Convolutional Neural Networks’, the data science blog. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Ημερομηνία πρόσβασης: 11-Ιανουαρίου-2018].
[31] Y. Bengio, P. Simard, και P. Frasconi, ‘Learning long-term dependencies with gradient descent is difficult’, IEEE Transactions on Neural Networks, τ. 5, τχ. 2, σσ 157–166, Μαρτίου 1994.
[32] S. Hochreiter και J. Schmidhuber, ‘Long Short-term Memory’, Neural computation, τ. 9, σσ 1735–80, Δεκεμβρίου 1997.
[33] ‘Understanding LSTM Networks -- colah’s blog’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Ημερομηνία πρόσβασης: 01-Φεβρουαρίου-2018].
[34] ‘Models built with TensorFlow’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://github.com/tensorflow/models/tree/master/tutorials/rnn/ptb. [Ημερομηνία πρόσβασης: 04-Φεβρουαρίου-2018].
[35] ‘Treebank-3 - Linguistic Data Consortium’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://catalog.ldc.upenn.edu/ldc99t42. [Ημερομηνία πρόσβασης: 04-Φεβρουαρίου-2018].
[36] W. Zaremba, I. Sutskever, και O. Vinyals, ‘Recurrent Neural Network Regularization’, arXiv:1409.2329 [cs], Σεπτεμβρίου 2014.
[37] ‘Speech and Language Processing’. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://web.stanford.edu/~jurafsky/slp3/. [Ημερομηνία πρόσβασης: 04-Φεβρουαρίου-2018].
[38] ‘Module: tf.contrib.rnn’, TensorFlow. [Έκδοση σε ψηφιακή μορφή]. Διαθέσιμο στο: https://www.tensorflow.org/api_docs/python/tf/contrib/rnn. [Ημερομηνία πρόσβασης: 04-Φεβρουαρίου-2018].