ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

ΑΝΑΓΝΩΡΙΣΗ ΤΗΣ ΣΗΜΑΣΙΑΣ ΣΥΜΒΟΛΩΝ ΣΕ ΜΑΘΗΜΑΤΙΚΑ ΚΕΙΜΕΝΑ ΜΕ ΧΡΗΣΗ ΤΕΧΝΙΚΩΝ ΜΗΧΑΝΙΚΗΣ ΜΑΘΗΣΗΣ

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΠΛΑΤΑΝΙΤΗ ΔΗΜΗΤΡΑΣ

Επιβλέπων: Παπασαλούρος Ανδρέας

Σάμος, Ιούλιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΙΓΑΙΟΥ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΜΑΘΗΜΑΤΙΚΩΝ

Αναγνώριση της σημασίας συμβόλων σε μαθηματικά κείμενα με χρήση τεχνικών μηχανικής μάθησης

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

της

ΠΛΑΤΑΝΙΤΗ ΔΗΜΗΤΡΑΣ

Επιβλέπων: Παπασαλούρος Ανδρέας

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή.

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
.....	.....	.....
Παπασαλούρος Α.	Τσολομύτης Α.	Κορνάρος Χ.

Σάμος, Ιούλιος 2022

Ευχαριστίες

Θα ήθελα να ευχαριστήσω, πρώτο από όλους, τον επιβλέποντα καθηγητή κ.Παπασαλούρο Ανδρέα για την εμπιστοσύνη που μου έδειξε με την ανάθεση της συγκεκριμένης διπλωματικής εργασίας, την καθοδήγηση που μου παρείχε και τη συνεισφορά του για την επιτέλεσή της. Στη συνέχεια, θα ήθελα να ευχαριστήσω τον καθηγητή κ.Τσολομύτη Αντώνιο για τη συμμετοχή του στην εργασία αυτή, με το να μας παράσχει την πρώτη εκδοχή των δεδομένων εκπαίδευσης που χρησιμοποιήθηκαν στους κώδικες που αναπτύχθηκαν. Τέλος, θα ήθελα να ευχαριστήσω τον κ.Τσολομύτη Αντώνιο και τον κ.Κορνάρο Χαράλαμπο που δέχτηκαν να είναι μέλη της τριμελούς επιτροπής αξιολόγησης της διπλωματικής εργασίας.

Περίληψη

Στη γλώσσα Braille, την οποία διαβάζουν τα άτομα με προβλήματα όρασης, τα σύμβολα για την τελεία και την υποδιαστολή είναι διαφορετικά. Στα μαθηματικά κείμενα το σύμβολο για τα δύο παραπάνω είναι το ίδιο (.). Δεδομένου ότι η εύρεση σαφών κανόνων για την αυτόματη αναγνώριση της σημασίας του συμβόλου της τελείας είναι δύσκολη, στο πλαίσιο αυτής της εργασίας, για την επίλυση του παραπάνω προβλήματος, μελετήθηκε η εφαρμογή τεχνικών μηχανικής μάθησης χρησιμοποιώντας επαναλαμβανόμενα νευρωνικά δίκτυα, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο.

Για την πραγμάτωση του σκοπού μας, αρχικά έγινε μία εισαγωγή σε βασικές έννοιες της μηχανικής μάθησης και των τεχνητών νευρωνικών δικτύων. Στη συνέχεια, παρουσιάστηκαν σε βάθος τα επαναλαμβανόμενα νευρωνικά δίκτυα, ο τρόπος λειτουργίας, οι τεχνικές εκπαίδευσης, οι επεκτάσεις καθώς και οι εφαρμογές τους σε διάφορους τομείς για τις σημερινές ανάγκες του ανθρώπου. Έπειτα, αναφέρεται το πρόβλημα που αντιμετωπίζεται στα επαναλαμβανόμενα νευρωνικά δίκτυα και η αντιμετώπισή του με χρήση των δικτύων μακράς βραχύχρονης μνήμης. Με εφόδιο αυτές τις γνώσεις αναπτύσσεται ο κώδικας ο οποίος ταξινομεί τα δεδομένα εισόδου, δηλαδή χαρακτήρες και σημεία στίξης, σε τρεις κατηγορίες: τελεία, υποδιαστολή και λοιπούς χαρακτήρες, με ακρίβεια που φτάνει το 93.33%. Τέλος, γίνεται ανάλυση του κώδικα αυτού και των αποτελεσμάτων που δίνει.

Λέξεις κλειδιά: Μηχανική μάθηση, τεχνητή νοημοσύνη, ταξινόμηση, εποπτευόμενη μάθηση, επαναλαμβανόμενα νευρωνικά δίκτυα, δίκτυα μακράς βραχύχρονης μνήμης.

Abstract

In Braille, the tactile writing system used by people who are visually impaired, the symbols for the period and the decimal point are different. In mathematical texts the symbol for the two above is the same (.). Since it is difficult to find clear rules for the automatic recognition of the meaning of the dot symbol, in order to solve this problem, in this diploma thesis we investigated the application of machine learning techniques using recurrent neural networks, both theoretically and in practical level.

For our purpose, at first, we made an introduction to the basic concepts of machine learning and artificial neural networks. Then, we made an in-depth presentation of recurrent neural networks and their mode of operation, training techniques, extensions as well as their applications in various fields for the current needs of man. In addition, we explored the problem encountered in recurrent neural networks and how to avoid it using long short term memory networks. Equipped with this knowledge, we developed the code which classifies the input data, i.e., characters and punctuation marks, into three categories: period, decimal point and other characters, with an accuracy of up to 93.33%. Finally, an attempt was made to analyze the code and its results.

Keywords: Machine learning, artificial intelligence, classification, supervised learning, recurrent neural networks, long short term networks.

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1

ΕΙΣΑΓΩΓΗ

Οι ηλεκτρονικοί υπολογιστές, πολλές φορές, ξεπερνούν κατά πολύ τον άνθρωπο όσον αφορά στην ταχύτητα και την ακρίβεια. Αυτό παρατηρείται, για παράδειγμα, στον τομέα της επιστήμης με τους αριθμητικούς υπολογισμούς και στις βιομηχανικές εφαρμογές όπου σχεδόν καμία διεργασία δεν διεκπεραιώνεται χωρίς αυτοματισμούς στις μηχανές. Ωστόσο, υπάρχουν διεργασίες οι οποίες δεν είναι καθόλου εύκολο για έναν υπολογιστή να τις φέρει εις πέρας, ενώ ο άνθρωπος μπορεί να τις καταφέρει χωρίς δυσκολία, όπως η αναγνώριση ενός αντικειμένου από την εικόνα του, η κατανόηση του προφορικού λόγου κτλ. Το κοινό χαρακτηριστικό αυτών των διεργασιών είναι ότι δεν υπάρχει ακριβής περιγραφή για το πώς ο άνθρωπος επιλύει αυτά τα προβλήματα. Σε αντίθεση, λοιπόν, με τον τομέα στον οποίο οι υπολογιστές χρησιμοποιούνται επιτυχώς, δεν υπάρχει ένα λεπτομερές και πλήρως προσδιορισμένο πρόγραμμα για την επίλυση. Ο άνθρωπος εφαρμόζει κάποιους κανόνες οι οποίοι είναι ασαφείς και ημιτελείς, καθώς και γνώσεις που τις απέκτησε εμπειρικά ή μέσω παραδειγμάτων. Ακριβής μαθηματική μοντελοποίηση αυτών φαίνεται αδύνατη ή, έστω, πολύ δύσκολη και χρονοβόρα. Αυτό ακριβώς το ζήτημα καλούμαστε να διερευνήσουμε μέσω της παρούσας διπλωματικής εργασίας, αναφορικά με το πρόβλημα της ταξινόμησης αντικειμένων.

1.1 Σκοπός της Διπλωματικής Εργασίας

Η ενασχόληση με τη συγγραφή της παρούσας διπλωματικής εργασίας προέκυψε μετά από την παρατήρηση της ολοένα και πιο ευρείας χρήσης των επαναλαμβανόμενων νευρωνικών δικτύων, μία κατηγορία νευρωνικών δικτύων πολλά υποσχόμενη, και των καλών αποτελεσμάτων που δίνουν. Σκοπός είναι η δημιουργία ενός προγράμματος στο οποίο θα εκπαιδεύεται ένα δίκτυο ώστε να είναι δυνατή η αυτόματη αναγνώριση της σημασίας του συμβόλου της τελείας σε μαθηματικά κείμενα γραμμένα στο TeX.

Η σκέψη που μας οδήγησε στο στόχο αυτόν ήρθε έπειτα από επαφή με τον κώδικα Braille, τον οποίο διαβάζουν τα άτομα με προβλήματα όρασης. Εκεί τα σύμβολα για την τελεία και την υποδιαστολή είναι διαφορετικά. Στα μαθηματικά κείμενα, όμως, το σύμβολο για τα δύο παραπάνω είναι το ίδιο (.). Λόγω του ότι η εύρεση σαφών κανόνων για την αυτόματη αναγνώριση της σημασίας του συμβόλου της τελείας είναι δύσκολη, μελετήθηκε η εφαρμογή τεχνικών μηχανικής μάθησης χρησιμοποιώντας επαναλαμβανόμενα νευρωνικά δίκτυα και επεκτάσεις τους, τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο.

Το πρώτο μέρος της εργασίας αποβλέπει στην παρουσίαση και την κατανόηση του θεωρητικού υπόβαθρου. Ειδικότερα, γίνεται μαθηματική περιγραφή των επαναλαμβανόμενων νευρωνικών δικτύων και αναλυτική αναφορά τόσο των τεχνικών εκπαίδευσής τους όσο και των επεκτάσεών τους. Επιπλέον, παρουσιάζονται σε βάθος τα δίκτυα μακράς βραχύχρονης μνήμης και οι παραλλαγές τους.

Στο δεύτερο μέρος μελετάται η εφαρμογή αυτών των τεχνικών μηχανικής μάθησης μέσα από την εκπαίδευση ενός δικτύου με χρήση της γλώσσας προγραμματισμού Python, με στόχο την αυτόματη αναγνώριση της σημασίας του συμβόλου της τελείας σε μαθηματικά κείμενα. Τέλος, αναφέρονται τα αποτελέσματα και τα συμπεράσματα που προκύπτουν.

1.2 Μηχανική Μάθηση

Πριν την ανάλυση των νευρωνικών δικτύων και των κατηγοριών τους, απαραίτητη είναι η αναφορά στον ακρογωνιαίο λίθο αυτών, τη μηχανική μάθηση (machine learning). Το ζήτημα της μαθηματικής μοντελοποίησης ασαφών κανόνων στον υπολογιστή αντιμετωπίζεται στον τομέα της μηχανικής μάθησης. Η μηχανή πρέπει να μπορεί να μάθει μία επιθυμητή

συμπεριφορά αν της δοθούν ατελείς πληροφορίες και παραδείγματα αντί για ένα ακριβές μοντέλο της κατάστασης, όπως, δηλαδή, ενεργεί και ο άνθρωπος. Το 1959, ο Arthur Samuel, πρωτοπόρος της Τεχνητής Νοημοσύνης, επινοώντας τον όρο «Μηχανική Μάθηση» τον όρισε ως το «πεδίο μελέτης που δίνει στους υπολογιστές την ικανότητα να μαθαίνουν, χωρίς ρητά να έχουν προγραμματιστεί». Ο Tom M. Mitchell πρότεινε έναν πιο επίσημο ορισμό που χρησιμοποιείται ευρέως: «Ένα πρόγραμμα υπολογιστή λέγεται ότι μαθαίνει από εμπειρία E ως προς μια κλάση εργασιών T και ένα μέτρο επίδοσης P, αν η επίδοσή του σε εργασίες της κλάσης T, όπως αποτιμάται από το μέτρο P, βελτιώνεται με την εμπειρία E».

Εικόνα 1: Η βασική διαφορά του παραδοσιακού προγραμματισμού με τη μηχανική μάθηση.[1]

Οι ερευνητές προσπάθησαν να προσεγγίσουν το ζήτημα με διάφορες μεθόδους, με ιδιαίτερα επιτυχή τα τεχνητά νευρωνικά δίκτυα. Τα τεχνητά νευρωνικά δίκτυα είναι δίκτυα από απλούς υπολογιστικούς κόμβους (νευρώνες) διασυνδεδεμένους μεταξύ τους. Είναι εμπνευσμένα από το Κεντρικό Νευρικό Σύστημα των έμβιων όντων, το οποίο προσπαθούν να προσομοιώσουν. Κάθε νευρώνας δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές, επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο. Ανάλογα με τη δομή των νευρώνων και τον τρόπο σύνδεσής τους, δημιουργούνται οι διάφορες μορφές των τεχνητών νευρωνικών δικτύων.

Στην εργασία αυτή θα ασχοληθούμε με τα επαναλαμβανόμενα νευρωνικά δίκτυα, μία κατηγορία νευρωνικών δικτύων πολλά υποσχόμενη, καθώς είναι η μοναδική με εσωτερική μνήμη. Αυτό το χαρακτηριστικό τους επιτρέπει να είναι πολύ ακριβή στις προβλέψεις τους στο τι ακολουθεί. Για αυτό το λόγο, προτιμώνται για ακολουθιακά δεδομένα, όπως ο λόγος, το κείμενο, ο καιρός κτλ. Αρχικά δημιουργήθηκαν τη δεκαετία του 1980, αλλά τα τελευταία χρόνια είδαμε τις πραγματικές δυνατότητές τους. Η αύξηση της υπολογιστικής ισχύος σε συνδυασμό με τον τεράστιο όγκο δεδομένων τον οποίο έχουμε να επεξεργαστούμε, καθώς και η δημιουργία των δικτύων μακράς βραχύχρονης μνήμης τη δεκαετία του 1990, έφεραν τα επαναλαμβανόμενα νευρωνικά δίκτυα στο προσκήνιο.

1.3 Κατηγοριοποίηση της Μηχανικής Μάθησης

Υπάρχουν τρία βασικά είδη μηχανικής μάθησης, ανάλογα με το είδος της εμπειρίας που επιτρέπεται στους αλγορίθμους να έχουν κατά τη διαδικασία της μάθησης. Η εποπτευόμενη (supervised), η μη εποπτευόμενη (unsupervised) και η ενισχυτική (reinforcement) μάθηση. Στην εργασία αυτή θα ασχοληθούμε κυρίως με την εποπτευόμενη μάθηση.

Στους περισσότερους αλγορίθμους επιτρέπεται η εμπειρία ενός ολόκληρου συνόλου δεδομένων (dataset). Ένα σύνολο δεδομένων είναι μία συλλογή πολλών παραδειγμάτων, τα οποία κάποιες φορές ονομάζονται σημεία δεδομένων (data points). Ένα από τα παλιότερα σύνολα δεδομένων που μελετήθηκε από στατιστικολόγους και ερευνητές μηχανικής μάθησης είναι το σύνολο δεδομένων Iris. Αποτελεί μία συλλογή από μετρήσεις διαφορετικών τμημάτων 150 φυτών ίριδας. Κάθε φυτό αντιστοιχεί σε ένα παράδειγμα. Τα χαρακτηριστικά μέσα σε

[1] Πηγή: https://www.geeksforgeeks.org/ml-machine-learning/ .

κάθε παράδειγμα είναι οι μετρήσεις κάθε τμήματος του φυτού, δηλαδή το μήκους του σέπαλου, το πλάτος του σέπαλου, το μήκος του πέταλου και το πλάτος του πέταλου. Το σύνολο δεδομένων, επίσης, έχει καταχωρημένο το είδος που ανήκει κάθε φυτό και περιλαμβάνει τρία είδη.

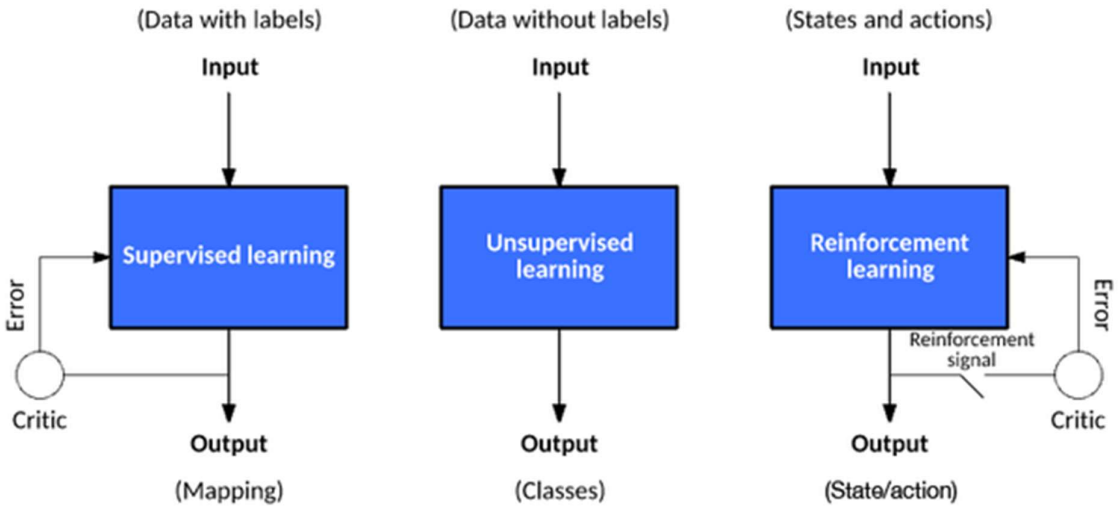
Στην εποπτευόμενη μάθηση ενός νευρωνικού δικτύου, η διαδικασία σχετίζεται με τα δεδομένα εκπαίδευσης, δηλαδή χρονικές σειρές εισόδου/εξόδου οι οποίες έχουν παρατηρηθεί εμπειρικά ή είναι τεχνητά κατασκευασμένες και αναπαριστούν παραδείγματα της επιθυμητής συμπεριφοράς του μοντέλου. Τα δεδομένα εκπαίδευσης χρησιμοποιούνται για να εκπαιδεύσουν ένα νευρωνικό δίκτυο έτσι ώστε αυτό λίγο-πολύ να αναπαράγει (προσαρμόσει) ακριβώς τα δεδομένα εκπαίδευσης και ελπίζοντας, έτσι, ότι το νευρωνικό δίκτυο θα κάνει το ίδιο γενικά με καινούργια δεδομένα εισόδου. Οι αλγόριθμοι εποπτευόμενης μάθησης αναλύουν τα δεδομένα εκπαίδευσης και παράγουν ένα μοντέλο το οποίο μπορεί να χρησιμοποιηθεί για να χαρακτηρίσει νέα παραδείγματα. Το βέλτιστο σενάριο επιτρέπει στον αλγόριθμο να καθορίσει σωστά την ετικέτα της κατηγορίας για άγνωστα μέχρι τώρα παραδείγματα. Με άλλα λόγια, όταν το εκπαιδευμένο νευρωνικό δίκτυο δέχεται μία ακολουθία ως είσοδο η οποία είναι κατά κάποιο τρόπο παρόμοια με την ακολουθία εκπαίδευσης που δέχτηκε ως είσοδο πιο πριν, τότε θα πρέπει να βγάλει μία έξοδο η οποία να μοιάζει με την έξοδο του αρχικού συστήματος. Αυτή η προσέγγιση πράγματι μοιάζει με τον τρόπο που μαθαίνει ο άνθρωπος υπό την επίβλεψη ενός δασκάλου: Ο δάσκαλος δίνει κατάλληλα παραδείγματα στον μαθητή για να απομνημονεύσει και ο μαθητής εξάγει γενικούς κανόνες από αυτά.

Για παράδειγμα, το σύνολο δεδομένων Iris περιλαμβάνει την αναφορά του είδους κάθε φυτού ίριδας. Ένας αλγόριθμος με εποπτευόμενη μάθηση μπορεί να μελετήσει αυτό το σύνολο δεδομένων και να μάθει να ταξινομεί τα φυτά ίριδες σε τρία διαφορετικά είδη με βάση τις μετρήσεις τους.

Ένα σημαντικότατο πρόβλημα που προκύπτει στην εποπτευόμενη μάθηση είναι η υπερπροσαρμογή (overfitting). Η υπερπροσαρμογή, η οποία θα αναλυθεί περεταίρω σε επόμενο κεφάλαιο, συμβαίνει όταν το κενό μεταξύ του σφάλματος εκπαίδευσης και του σφάλματος επικύρωσης είναι πολύ μεγάλο. Δηλαδή, όταν κατά την εκπαίδευση το σφάλμα είναι μικρό, αλλά μετά την ολοκλήρωσή της το σφάλμα επικύρωσης είναι πολύ μεγάλο.

Στη μη εποπτευόμενη μάθηση, τα δεδομένα εκπαίδευσης που χρησιμοποιούνται είναι μόνο τα δεδομένα εισόδου. Στόχος είναι η συσχέτιση και ο προσδιορισμός μοτίβων στα δεδομένα αυτά χωρίς τη βοήθεια των δεδομένων εξόδου. Ως είδος μάθησης μοιάζει με τις μεθόδους που χρησιμοποιεί ο άνθρωπος για να συμπεράνει ότι κάποια αντικείμενα ή γεγονότα ανήκουν στην ίδια τάξη, όπως όταν παρατηρεί το βαθμό ομοιότητας μεταξύ αντικειμένων.

Στην ενισχυτική μάθηση, όπως και στη μη εποπτευόμενη, εισάγονται στον αλγόριθμο παραδείγματα χωρίς ετικέτες. Όμως, για τα παραδείγματα αυτά γίνεται θετική ή αρνητική ανατροφοδότηση και το σύστημα «επιβραβεύεται» ή «τιμωρείται», αντίστοιχα. Με αυτό τον τρόπο, ο αλγόριθμος μαθαίνει να επιλέγει την πιο κατάλληλη ενέργεια για το επόμενο βήμα. Με τον ανθρώπινο τρόπο σκέψης, δηλαδή, μαθαίνει μέσα από δοκιμές και λάθη. Τα λάθη βοηθούν τον άνθρωπο να μάθει, επειδή έχουν συνέπειες (κόστος, χάσιμο χρόνου κτλ.) οι οποίες αποδεικνύουν ότι κάποια ενέργεια είναι λιγότερο πιθανό να πετύχει από άλλες. Ένα παράδειγμα ενισχυτικής μάθησης, είναι όταν οι ηλεκτρονικοί υπολογιστές μαθαίνουν να παίζουν ηλεκτρονικά παιχνίδια μόνοι τους.



Εικόνα 2: Τρία μοντέλα μάθησης για αλγόριθμους νευρωνικών δικτύων.[2]

Μία άλλη κατηγοριοποίηση της μηχανικής μάθησης έχει να κάνει με το επιθυμητό εξαγόμενο αποτέλεσμα ενός εκπαιδευμένου συστήματος και περιλαμβάνει τις εξής κατηγορίες:

- Ταξινόμηση (Classification): Όταν τα δεδομένα εισόδου είναι χωρισμένα σε δύο ή περισσότερες κατηγορίες και το σύστημα πρέπει να παραγάγει ένα μοντέλο το οποίο να αντιστοιχεί άγνωστα δεδομένα εισόδου σε μία ή περισσότερες από αυτές τις κατηγορίες. Αυτό κατά κανόνα αντιμετωπίζεται με εποπτευόμενη μάθηση. Ένα παράδειγμα ταξινόμησης είναι το φιλτράρισμα της ανεπιθύμητης αλληλογραφίας (spam), όπου τα δεδομένα εισόδου είναι email και οι κατηγορίες είναι «spam» και «όχι spam».
- Παλινδρόμηση (Regression): Όταν τα εξαγόμενα αποτελέσματα είναι συνεχή και όχι διακριτά. Εδώ, επίσης, χρησιμοποιείται εποπτευόμενη μάθηση.
- Συσταδοποίηση (Clustering): Όταν ένα σετ δεδομένων εισόδου πρόκειται να χωριστεί σε ομάδες. Σε αντίθεση με την ταξινόμηση, οι ομάδες δεν είναι γνωστές εκ των προτέρων και, επομένως, συνήθως χρησιμοποιείται μη εποπτευόμενη μάθηση.

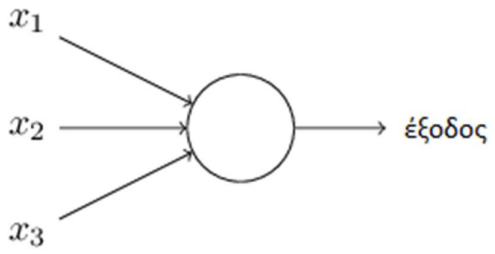
[2] Πηγή: https://developer.ibm.com/articles/cc-models-machine-learning/ .

ΚΕΦΑΛΑΙΟ 2

ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ

Περί το 1950, ο Frank Rosenblatt, ψυχολόγος ο οποίος συχνά καλείται «πατέρας της βαθιάς μάθησης», εμπνευσμένος από τους Warren McCulloch και Walter Pitts, διεξήγαγε τις πρώτες εργασίες για τα perceptrons, έναν τύπο τεχνητού νευρώνα, με αποκορύφωμα την κατασκευή του Mark I Perceptron το 1960. Αυτός ήταν ουσιαστικά ο πρώτος υπολογιστής που μπορούσε να μάθει νέες δεξιότητες κάνοντας δοκιμές και λάθη (trial and error), χρησιμοποιώντας έναν τύπο νευρωνικού δικτύου που προσομοιώνει την ανθρώπινη σκέψη. Σήμερα, είναι πιο συνήθης η χρήση άλλων μοντέλων τεχνητών νευρώνων, αλλά για την κατανόηση αυτών είναι καλό να γνωρίζουμε πώς λειτουργούν τα perceptrons.

Ένα perceptron δέχεται κάποια δυαδικά δεδομένα εισόδου x_1, x_2, ..., και παράγει μία μοναδική δυαδική έξοδο:



Εικόνα 3: Perceptron με τρία δεδομένα εισόδου.[3]

Ένας απλός κανόνας υπολογισμού της εξόδου είναι με τη χρήση βαρών (weights), w_1, w_2, ..., δηλαδή πραγματικών αριθμών που εκφράζουν τη βαρύτητα των αντίστοιχων δεδομένων εισόδου για την παραγωγή της εξόδου. Η έξοδος του νευρώνα καθορίζεται από το αν το βεβαρημένο άθροισμα του γινομένου των δεδομένων εισόδου με τα αντίστοιχα βάρη είναι μικρότερο ή μεγαλύτερο από μία τιμή κατωφλίου. Η τιμή κατωφλίου είναι πραγματικός αριθμός ο οποίος είναι μία παράμετρος του νευρώνα. Αυτό γράφεται αλγεβρικά ως εξής:

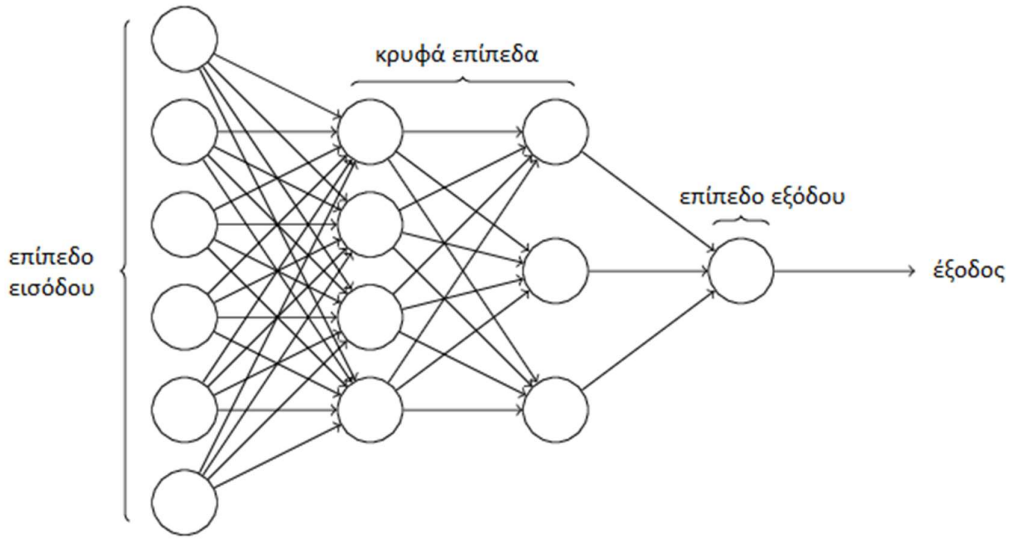
$$\text{έξοδος} = \begin{cases} 0 \text{ αν } \sum_j w_j x_j \leq \text{κατώφλι} \\ 1 \text{ αν } \sum_j w_j x_j > \text{κατώφλι} \end{cases}$$

Προκειμένου να απλοποιηθούν αυτοί οι τύποι, γράφουμε το $\sum_j w_j x_j$ ως $w \cdot x$, όπου w και x είναι διανύσματα των οποίων τα στοιχεία είναι τα βάρη και τα δεδομένα εισόδου αντίστοιχα. Επιπλέον, μετακινούμε το κατώφλι στο άλλο μέλος της ανισότητας και το αντικαθιστούμε με τη λεγόμενη πόλωση perceptron, $b \equiv -\text{κατώφλι}$. Η πόλωση (bias) μετρά πόσο εύκολο είναι για το perceptron να εξάγει ως αποτέλεσμα 1. Αν η πόλωση είναι πολύ μεγάλη, τότε είναι εξαιρετικά εύκολο για το perceptron να εξάγει ως αποτέλεσμα 1. Οπότε προκύπτει:

$$\text{έξοδος} = \begin{cases} 0 \text{ αν } w \cdot x + b \leq 0 \\ 1 \text{ αν } w \cdot x + b > 0 \end{cases}$$

Όσον αφορά στην αρχιτεκτονική των νευρωνικών δικτύων που χρησιμοποιούνται στις μέρες μας, ας υποθέσουμε ότι έχουμε το εξής δίκτυο:

[3] Πηγή: http://neuralnetworksanddeeplearning.com/chap1.html .



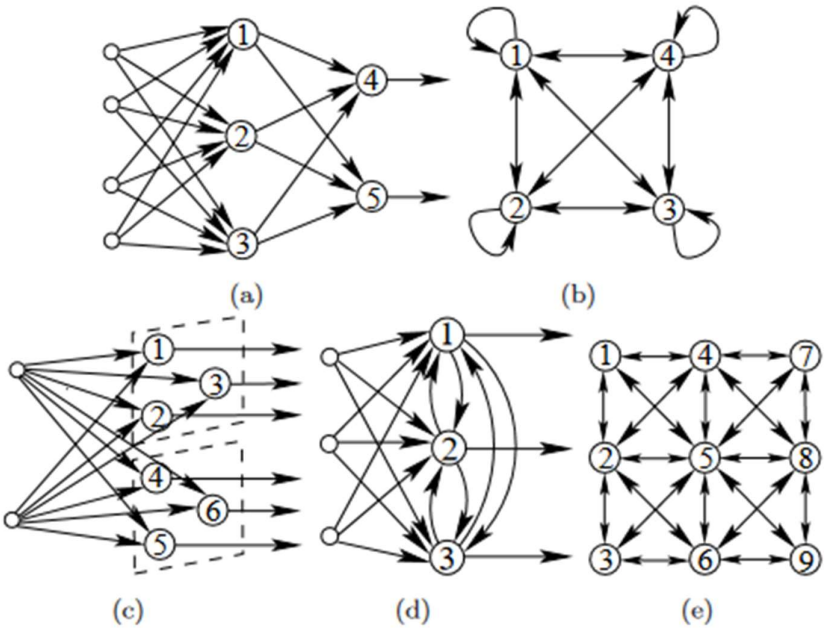
Εικόνα 4: Η αρχιτεκτονική των νευρωνικών δικτύων.[4]

Το δίκτυο αυτό περιλαμβάνει 4 επίπεδα: το επίπεδο εισόδου στα αριστερά, έπειτα δύο κρυφά επίπεδα και, τέλος, το επίπεδο εξόδου. Το επίπεδο εισόδου περιλαμβάνει τόσους νευρώνες εισόδου όσα και τα δεδομένα εισόδου. Ο αριθμός των κρυφών επιπέδων σε ένα δίκτυο, η ονομασία των οποίων σημαίνει απλά ότι δεν είναι ούτε είσοδος ούτε έξοδος του δικτύου, μπορεί να κυμανθεί από 1 έως πολλαπλά. Το επίπεδο εξόδου παραπάνω περιλαμβάνει έναν νευρώνα εξόδου, άλλα σε άλλα δίκτυα μπορεί να περιλαμβάνει περισσότερους.

Για ιστορικούς λόγους, τέτοια δίκτυα πολλαπλών επιπέδων μερικές φορές καλούνται πολυεπίπεδα perceptrons (multilayer perceptrons ή MLPs), παρόλο που αποτελούνται από σιγμοειδείς νευρώνες και όχι perceptrons.

Χαρακτηριστικό γνώρισμα για ένα νευρωνικό δίκτυο αποτελεί η αρχιτεκτονική του, τα χαρακτηριστικά των κόμβων του και οι κανόνες μάθησης. Η αρχιτεκτονική του δικτύου αναπαρίσταται από τον πίνακα των βαρών $W = [\, w_{ij} \,]$, όπου w_{ij} είναι το βάρος που συνδέει τον κόμβο i με τον κόμβο j. Όταν $w_{ij} = 0$, δεν υπάρχει σύνδεση από τον κόμβο i στον κόμβο j. Θέτοντας κάποια w_{ij} μηδέν, μπορούν να προκύψουν διαφορετικές τοπολογίες δικτύου. Τα νευρωνικά δίκτυα μπορούν σε γενικές γραμμές να χωριστούν σε δίκτυα πρόσθιας τροφοδότησης (feedforward networks), επαναλαμβανόμενα (recurrent neural networks) και τα υβριδικά αυτών. Δημοφιλείς τοπολογίες δικτύου είναι τα πλήρως συνδεδεμένα πολυεπίπεδα πρόσθιας τροφοδότησης δίκτυα (fully connected layered feedforward networks), τα επαναλαμβανόμενα δίκτυα, τα δίκτυα πλέγματος (lattice networks), τα πολυεπίπεδα πρόσθιας τροφοδότησης δίκτυα με πλευρικές συνδέσεις (layered feedforward networks with lateral connections) και τα κυψελωτά δίκτυα (cellular networks).

[4] Πηγή: http://neuralnetworksanddeeplearning.com/chap1.html .



Εικόνα 5: (a) Layered feedforward network, (b) Recurrent network, (c) Lattice network 2 διαστάσεων, (d) Layered feedforward networks with lateral connections, (e) Cellular network. Οι κύκλοι με τους αριθμούς αναπαριστούν τους νευρώνες, ενώ οι μικροί τους κόμβους δεδομένων εισόδου.[5]

- Σε ένα δίκτυο πρόσθιας τροφοδότησης (feedforward network), οι συνδέσεις μεταξύ των νευρώνων είναι προς μία κατεύθυνση. Ένα δίκτυο πρόσθιας τροφοδότησης συνήθως είναι κατασκευασμένο σε επίπεδα. Σε ένα τέτοιο δίκτυο, δεν υπάρχει σύνδεση μεταξύ των νευρώνων του ίδιου επιπέδου, αλλά ούτε και κάποια ανάδραση μεταξύ των επιπέδων. Σε ένα πλήρως συνδεδεμένο πολυεπίπεδο δίκτυο πρόσθιας τροφοδότησης, κάθε κόμβος σε κάθε επίπεδο είναι συνδεδεμένος με κάθε κόμβο στο επόμενο επίπεδο. Το πολυεπίπεδο perceptron (MLP) δίκτυο που είδαμε παραπάνω είναι πλήρως συνδεδεμένο πολυεπίπεδο δίκτυο πρόσθιας τροφοδότησης.
- Σε ένα επαναλαμβανόμενο δίκτυο, υπάρχει τουλάχιστον μία ανάδραση. Με τα δίκτυα αυτά θα ασχοληθούμε σε αυτή την εργασία.
- Ένα δίκτυο πλέγματος αποτελείται από έναν ή περισσότερους πίνακες νευρώνων. Κάθε πίνακας έχει το αντίστοιχο δικό του σετ από κόμβους δεδομένων εισόδου.
- Ένα πολυεπίπεδο δίκτυο πρόσθιας τροφοδότησης με πλευρικές συνδέσεις έχει πλευρικές συνδέσεις μεταξύ των νευρώνων στο ίδιο επίπεδο.
- Ένα κυψελωτό δίκτυο αποτελείται από ισαπέχοντες νευρώνες, οι οποίοι ονομάζονται κελιά (cells), που επικοινωνούν μόνο με τους νευρώνες στην άμεση «γειτονιά» τους. Τα γειτονικά κελιά αλληλοσυνδέονται και, έτσι, κάθε κελί «ξυπνά» από τα δικά του σήματα και από τα σήματα που έρχονται από το γειτονικό του κελί.

2.1 Είδη Συνάρτησης Ενεργοποίησης

Πριν από την ανάλυση των ειδών ενεργοποίησης που θα γίνει σε αυτή την ενότητα, ιδιαίτερα σημαντική είναι η ερμηνεία κάποιων όρων βασικών για τη μηχανική μάθηση. Οι όροι αυτοί είναι οι εξής:

Σύνολο δεδομένων εκπαίδευσης (training dataset)

Δεδομένα εκπαίδευσης ονομάζονται τα αρχικά δεδομένα τα οποία χρησιμοποιούνται με σκοπό την εκπαίδευση μοντέλων μηχανικής μάθησης. Εισάγονται στο δίκτυο και αυτό

[5] Πηγή: https://www.researchgate.net/publication/278654277_Neural_Networks_and_Statistical_Learning .

καλείται να τα μάθει ερχόμενο κατ' επανάληψη σε επαφή μαζί τους. Το δίκτυο εκπαιδεύεται καλύτερα όταν υπάρχει μεγάλος όγκος δεδομένων εκπαίδευσης.

Σύνολο δεδομένων επικύρωσης (test dataset)
Σύνολο δεδομένων επικύρωσης ονομάζεται ένα σύνολο παραδειγμάτων το οποίο χρησιμοποιείται με σκοπό την αξιολόγηση της απόδοσης ενός μοντέλου μηχανικής μάθησης. Το σύνολο αυτό είναι ανεξάρτητο του συνόλου των δεδομένων εκπαίδευσης και, έτσι, το δίκτυο καλείται να αναγνωρίσει τα δεδομένα του.

Εποχή (epoch)
Ο αριθμός των εποχών δηλώνει τον αριθμό των επαναλήψεων (ή, απλούστερα, των «περασμάτων») σε ολόκληρο το σύνολο δεδομένων εκπαίδευσης που έχει ολοκληρώσει ο αλγόριθμος της μηχανικής μάθησης, ώστε να εκπαιδευτεί.

Παρτίδα (batch)
Τα σύνολα δεδομένων συνήθως χωρίζονται σε παρτίδες, ειδικά αν η ποσότητα των δεδομένων είναι πολύ μεγάλη. Το μέγεθος των παρτίδων δηλώνει τον αριθμό των παραδειγμάτων από τα δεδομένα εκπαίδευσης τα οποία χρησιμοποιούνται σε μία επανάληψη. Αν το μέγεθος των παρτίδων ισούται με τον αριθμό των παραδειγμάτων ολόκληρου του συνόλου των δεδομένων εκπαίδευσης, τότε η κατάσταση λειτουργίας ονομάζεται batch mode. Αν το μέγεθος των παρτίδων είναι μεγαλύτερο από 1, αλλά μικρότερο από τον αριθμό των παραδειγμάτων ολόκληρου του συνόλου των δεδομένων εκπαίδευσης, τότε η κατάσταση λειτουργίας ονομάζεται mini-batch mode. Τέλος, αν το μέγεθος των παρτίδων ισούται με 1, δηλαδή η κλίση (gradient) και οι παράμετροι του νευρωνικού δικτύου ενημερώνονται για κάθε δείγμα του συνόλου δεδομένων, τότε η κατάσταση λειτουργίας ονομάζεται stochastic mode.

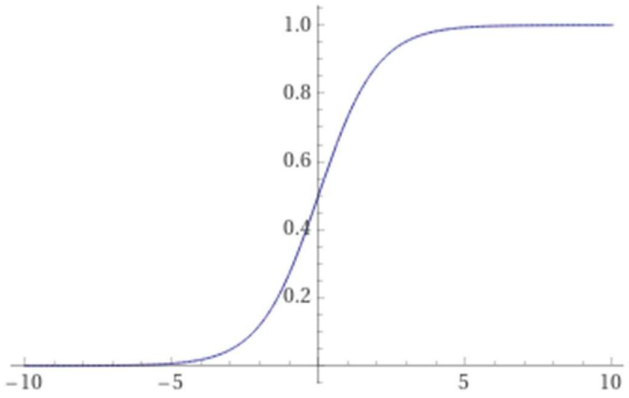
Ρυθμός εκμάθησης (learning rate)
Ο ρυθμός εκμάθησης είναι μία υπερπαράμετρος ενός αλγορίθμου βελτιστοποίησης, η οποία καθορίζει το μέγεθος του βήματος σε κάθε επανάληψη, καθώς κινείται προς μία ελάχιστη τιμή της συνάρτησης απώλειας. Δεδομένου ότι επηρεάζει σε ποιο βαθμό οι νεοαποκτηθείσες πληροφορίες υπερισχύουν των παλιών πληροφοριών, μεταφορικά αντιπροσωπεύει την ταχύτητα με την οποία εκπαιδεύεται το δίκτυο. Για παράδειγμα, ένας μικρότερος ρυθμός εκμάθησης σημαίνει ότι το μοντέλο αλλάζει τις τιμές των βαρών σε μικρότερη κλίμακα, ενώ ένας μεγαλύτερος ρυθμός εκμάθησης κάνει το αντίθετο, για κάθε χρονικό βήμα.

Υπερπαράμετρος (hyperparameter)
Υπερπαράμετρος ονομάζεται μία παράμετρος, η τιμή της οποίας χρησιμοποιείται για τον έλεγχο της διαδικασίας της μάθησης. Αντίθετα, οι τιμές των άλλων παραμέτρων, δηλαδή των βαρών και των πολώσεων, προκύπτουν κατά τη διάρκεια της εκπαίδευσης. Παραδείγματα υπερπαραμέτρων είναι ο ρυθμός εκμάθησης, το μέγεθος των παρτίδων και ο αριθμός των εποχών. Αξίζει να σημειωθεί ότι στην περίπτωση που δε δοθούν στις υπερπαραμέτρους κατάλληλες τιμές, το δίκτυο είναι πιθανό να μην εξάγει τα επιθυμητά αποτελέσματα.

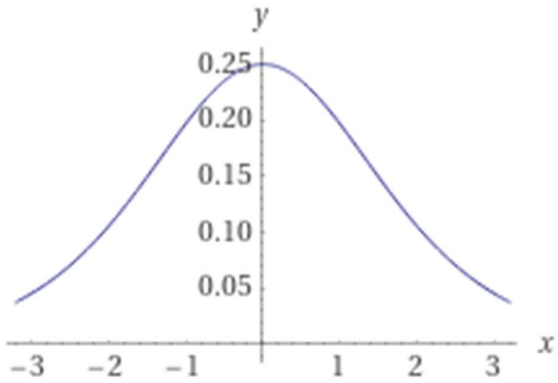
Με εφόδιο τις γνώσεις αυτές, η μελέτη των συναρτήσεων ενεργοποίησης θα γίνει ευκολότερα. Μία συνάρτηση ενεργοποίησης (activation function) παίρνει έναν αριθμό από τα δεδομένα εισόδου και εκτελεί μία συγκεκριμένη μαθηματική πράξη σε αυτόν. Μερικές από αυτές είναι οι εξής:

Σιγμοειδής (Sigmoid)
Τύπος: $\sigma(x) = \frac{1}{1+e^{-x}}$.

Γραφική παράσταση:



Εικόνα 6: Γραφική παράσταση σιγμοειδούς συνάρτησης.[6]



Εικόνα 7: Γραφική παράσταση της παραγώγου της σιγμοειδούς συνάρτησης. Η παράγωγος τείνει στο 0 και στις δύο πλευρές. Τείνει να γίνει μία ευθεία γραμμή. Αυτό σημαίνει ότι οι αντίστοιχοι νευρώνες παθαίνουν κορεσμό.[7]

Η σιγμοειδής συνάρτηση παίρνει έναν αριθμό με πραγματική τιμή και τον μετασχηματίζει ώστε να ανήκει στο διάστημα [0,1]. Συγκεκριμένα, οι αριθμοί με πολύ χαμηλή αρνητική τιμή μετασχηματίζονται σε 0, ενώ οι αριθμοί με πολύ μεγάλη θετική τιμή σε 1. Παλιότερα, η σιγμοειδής συνάρτηση χρησιμοποιούνταν συχνά, επειδή αποτελεί μία καλή προσέγγιση στην πυροδότηση ενός νευρώνα: από το να μην πυροδοτηθεί καθόλου (0) στο να πυροδοτηθεί πλήρως με αποτέλεσμα τον κορεσμό (1). Τον τελευταίο καιρό, η σιγμοειδής συνάρτηση χρησιμοποιείται σπάνια, αφού έχει σημαντικά μειονεκτήματα:

- Μπορεί να προκαλέσει κορεσμό και να «σκοτώσει» την κλίση. Κορεσμό παθαίνουν οι νευρώνες όταν οι τιμές τους είναι πολύ κοντά στις οριακές τιμές της συνάρτησης, στην περίπτωση αυτή, δηλαδή, το 0 και το 1. Αυτό συνεπάγεται ότι η κλίση είναι μηδενική ή σχεδόν μηδενική και άρα γραφικά αναπαρίσταται ως ευθεία γραμμή. Οπότε, μετά τη διαδικασία της οπισθοδιάδοσης (backpropagation) δε θα είναι δυνατό να εξαχθεί κάποια τιμή από τον νευρώνα στα βάρη. Επιπλέον, τα βάρη πρέπει να αρχικοποιηθούν με μεγάλη προσοχή για να αποφευχθεί ο κορεσμός. Για παράδειγμα, αν τα αρχικά βάρη είναι πολύ μεγάλα, τότε οι περισσότεροι νευρώνες θα πάθουν κορεσμό και το δίκτυο μετά βίας θα εκπαιδευτεί.
- Οι έξοδοι του νευρώνα δεν είναι κεντραρισμένες γύρω από το μηδέν. Αυτό δεν είναι επιθυμητό, αφού νευρώνες σε μεταγενέστερα επίπεδα της επεξεργασίας σε ένα νευρωνικό δίκτυο θα λαμβάνουν δεδομένα τα οποία δεν θα είναι κεντραρισμένα γύρω

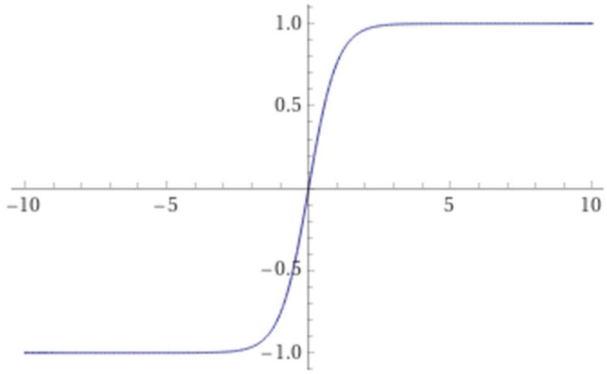
[6] Η γραφική παράσταση έγινε με τη βοήθεια του https://www.wolframalpha.com/ .
[7] Η γραφική παράσταση έγινε με τη βοήθεια του https://www.wolframalpha.com/ .

από το μηδέν. Αυτό θα έχει επιπτώσεις κατά τη διάρκεια της κατάβασης κλίσης (gradient descent), επειδή αν τα δεδομένα εισόδου σε έναν νευρώνα είναι πάντα θετικά, τότε η κλίση στα βάρη κατά τη διάρκεια της οπισθοδιάδοσης θα γίνει εξ ολοκλήρου ή θετική ή αρνητική, το οποίο θα προκαλούσε ένα μη επιθυμητό ζιγκ-ζαγκ στις ενημερώσεις της κλίσης για τα βάρη. Ωστόσο, παρατηρούμε ότι όταν οι κλίσεις αυτές προστεθούν σε μία παρτίδα δεδομένων, η τελική ανανέωση για τα βάρη μπορεί να έχει μεταβλητά πρόσημα, κάτι που μετριάζει το πρόβλημα. Συνεπώς, αυτό είναι μία ταλαιπωρία, αλλά έχει λιγότερο σοβαρές συνέπειες συγκριτικά με το παραπάνω πρόβλημα κορεσμένης ενεργοποίησης.

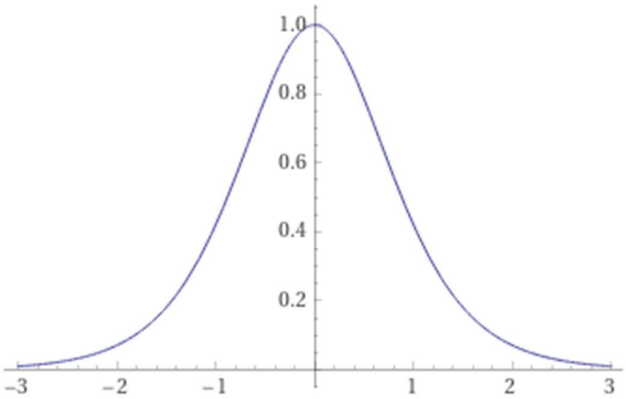
Υπερβολική εφαπτομένη (tanh)

Τύπος: $\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1}$.

Γραφική παράσταση:



Εικόνα 8: Γραφική παράσταση υπερβολικής εφαπτομένης.[8]



Εικόνα 9: Γραφική παράσταση της παραγώγου της υπερβολικής εφαπτομένης. Η παράγωγος τείνει στο 0 και στις δύο πλευρές. Συνεπώς, ισχύουν τα ίδια με τη σιγμοειδή συνάρτηση.[9]

Η υπερβολική εφαπτομένη παίρνει έναν αριθμό με πραγματική τιμή και τον μετασχηματίζει ώστε να ανήκει στο διάστημα [-1,1]. Όπως και η σιγμοειδής συνάρτηση, μπορεί να προκαλέσει κορεσμό στους νευρώνες. Αντίθετα, ωστόσο, από τη σιγμοειδή συνάρτηση, οι έξοδοι της

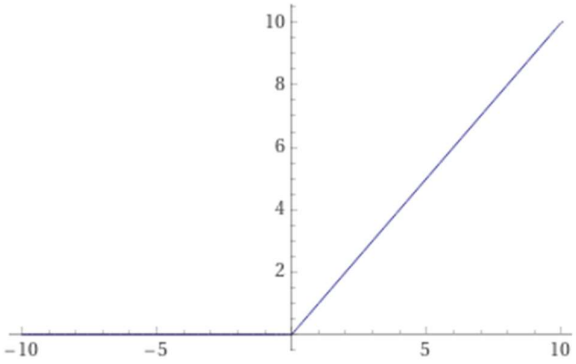
[8] Η γραφική παράσταση έγινε με τη βοήθεια του https://www.wolframalpha.com/ .
[9] Η γραφική παράσταση έγινε με τη βοήθεια του https://www.wolframalpha.com/ .

υπερβολικής εφαπτομένης είναι κεντραρισμένες γύρω από το μηδέν. Επομένως, στην πράξη η υπερβολική εφαπτομένη είναι πάντα προτιμότερη της σιγμοειδούς.

ReLU

Τύπος: $f(x) = \max(0, x)$.

Γραφική παράσταση:



Εικόνα 10: Γραφική παράσταση ReLU. (Η παράγωγος της ReLU είναι μία σταθερά, 0 ή 1, επομένως σπανιότερα προκύπτει πρόβλημα της εξαφάνισης των gradient.)[10]

Η Rectified Linear Unit (ReLU) έχει γίνει πολύ δημοφιλής τα τελευταία χρόνια. Αυτό που κάνει είναι το ότι οριοθετεί την ενεργοποίηση στο μηδέν. Η χρήση της έχει διάφορα πλεονεκτήματα και μειονεκτήματα:

- Παρατηρήθηκε ότι επιταχύνει σε μεγάλο βαθμό τη σύγκλιση της στοχαστικής κλίσης κατάβασης (stochastic gradient descent) συγκριτικά με τις δύο προηγούμενες συναρτήσεις. Υποστηρίζεται, χωρίς να είναι ακόμα σίγουρο, ότι αυτό οφείλεται στη γραμμική μορφή της η οποία δεν προκαλεί κορεσμό.
- Σε σύγκριση με τις δύο προηγούμενες συναρτήσεις οι οποίες είναι υπολογιστικά ακριβές λόγω των εκθετικών συναρτήσεων, η ReLU μπορεί να εφαρμοστεί απλά οριοθετώντας έναν πίνακα ενεργοποιήσεων στο μηδέν.
- Δυστυχώς, οι μονάδες ReLU μπορεί να είναι εύθραυστες κατά τη διάρκεια της εκπαίδευσης και μπορεί να «πεθάνουν». Για παράδειγμα, μια μεγάλη κλίση που ρέει διά μέσου ενός νευρώνα ReLU θα μπορούσε να προκαλέσει την ενημέρωση των βαρών με τέτοιο τρόπο ώστε ο νευρώνας να μην ενεργοποιηθεί ποτέ ξανά σε κανένα σημείο δεδομένων. Αν συμβεί αυτό, τότε η κλίση που ρέει διά μέσου της μονάδας θα είναι για πάντα μηδέν από εκείνο το σημείο και μετά. Με άλλα λόγια, οι μονάδες ReLU μπορούν να «πεθάνουν» αμετάκλητα κατά τη διάρκεια της εκπαίδευσης, καθώς μπορούν να «πεταχτούν» από την πολλαπλότητα των δεδομένων. Για παράδειγμα, ενδέχεται να διαπιστώσουμε ότι έως και το 40% του δικτύου μας μπορεί να είναι "νεκρό" (δηλαδή νευρώνες που δεν ενεργοποιούνται ποτέ σε ολόκληρο το σύνολο δεδομένων εκπαίδευσης) αν ο ρυθμός εκμάθησης είναι πολύ υψηλός. Με μια σωστή ρύθμιση του ρυθμού εκμάθησης, αυτό αποτελεί σπανιότερα πρόβλημα.

Ολοκληρώνοντας για τα είδη των πιο συνηθισμένων νευρώνων και τις συναρτήσεις ενεργοποίησής τους, αξίζει να σημειωθεί ότι είναι πολύ σπάνιος ο συνδυασμός διαφορετικών ειδών νευρώνων στο ίδιο δίκτυο, αν και δεν υπάρχει κάποιο πρόβλημα μεγάλης σημασίας που να προκύπτει από αυτό.

[10] Η γραφική παράσταση έγινε με τη βοήθεια του https://www.wolframalpha.com/ .

2.2 Συνάρτηση Απώλειας

Το αποτέλεσμα που προκύπτει από τις συναρτήσεις ενεργοποίησης είναι αναγκαίο να αξιολογηθεί από τον προγραμματιστή που υλοποιεί το δίκτυο, ώστε να κρίνει αν είναι ακριβές. Αυτό γίνεται με τη βοήθεια της συνάρτησης απώλειας (loss function). Η συνάρτηση απώλειας μετρά πόσο καλό είναι το μοντέλο όσον αφορά στην πρόβλεψη του αναμενόμενου αποτελέσματος. Αν οι προβλέψεις είναι λανθασμένες, η συνάρτηση απώλειας θα δώσει έναν υψηλό αριθμό, ενώ αν οι προβλέψεις είναι σωστές, θα δώσει ένα χαμηλό αριθμό. Ο τύπος του μέσου όρου των συναρτήσεων απώλειας είναι ο εξής:

$$L = \frac{1}{N}\sum_i L_i(f(x_i, W), y_i),$$

όπου x_i είναι η εικόνα εισόδου, y_i είναι ο ακέραιος αριθμός ο οποίος αντιστοιχεί στην πραγματική κλάση και i είναι ο αριθμός των κλάσεων της ταξινόμησης και κυμαίνεται μεταξύ του 1 και του N.

Γνωστές συναρτήσεις απώλειας είναι η SVM Loss, η Softmax Classifier, η Logistic Regression κτλ. Στην ενότητα αυτή θα αναλυθούν δύο βασικές από αυτές, η SVM Loss και η Softmax.

SVM Loss

Η SVM Loss αθροίζει τις λανθασμένες κατηγορίες και συγκρίνει τα αποτελέσματα με αυτά της σωστής κλάσης. Αν ο συνολικός αριθμός που πετυχαίνει η σωστή κλάση είναι μεγαλύτερος από το συνολικό αριθμό των λανθασμένων κατηγοριών συν κάποιο περιθώριο, δηλαδή τη μονάδα, τότε επιτυγχάνεται μηδενική απώλεια. Η SVM Loss ορίζεται ως εξής:

$$L_i = \sum_{j\neq y_i} \begin{cases} 0, \alpha\nu\ s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1, \alpha\lambda\lambda\iota\acute{\omega}\varsigma \end{cases} \text{ ή } L_i = \sum_{j\neq y_i} \max\left(0, s_j - s_{y_i} + 1\right), 0 < L_i < \infty,$$

όπου s είναι το $f(x_i, W)$, s_j είναι ο προβλεπόμενος συνολικός αριθμός για την κλάση, ο οποίος προέρχεται από τον ταξινομητή και s_{y_i} είναι ο συνολικός αριθμός της πραγματικής κλάσης.

Softmax Classifier

Η Softmax Classifier θεωρείται μία τεχνική πολλαπλής παλινδρόμησης (multiple regression). Κάθε κλάση αναπαρίσταται ως μία πιθανότητα με εύρος τιμών από 0 έως και 1 και συνολικό άθροισμα πιθανοτήτων 1. Ο τύπος της είναι ο εξής:

$$P(Y = k|X = x_i) = \frac{e^s}{\sum_j e^{s_j}},$$

όπου s είναι το $f(x_i, W)$. Το κλάσμα $\frac{e^s}{\sum_j e^{s_j}}$ είναι η συνάρτηση Softmax.

Η Softmax Classifier χρησιμοποιεί την απώλεια της διασταυρούμενης εντροπίας. Η Softmax Classifier παίρνει το όνομά της από τη συνάρτηση Softmax, η οποία εξάγει ως αποτέλεσμα την πιθανότητα για κάθε κλάση και αυτές οι πιθανότητες αθροίζονται όλες μαζί σε 1, έτσι ώστε να μπορεί να εφαρμοστεί η διασταυρούμενη εντροπία. Η διασταυρούμενη εντροπία είναι το άθροισμα των αρνητικών λογαρίθμων των πιθανοτήτων και περαιτέρω ανάλυσή της θα γίνει παρακάτω.

Η συνάρτηση Softmax Loss έχει τύπο τον εξής:

$$L_i = -logP((Y = y_i|X = x_i).$$

Συμπερασματικά, για την καλή απόδοση ενός δικτύου, ιδιαίτερα σημαντική είναι η μείωση των αποτελεσμάτων της συνάρτησης απώλειας, αφού, όπως περιγράφτηκε, ο ρόλος της είναι να κρίνει κατά πόσο τα αποτελέσματα της προσέγγισης είναι τα επιθυμητά.

2.3 Συνάρτηση Κόστους

Σε ένα δίκτυο, στο επίπεδο πριν το επίπεδο εξόδου, κάθε νευρώνας φέρει μία συνάρτηση απώλειας. Για το τελικό αποτέλεσμα το οποίο προκύπτει στο επίπεδο εξόδου, θεωρείται αναγκαία η συνάρτηση κόστους. Συνάρτηση κόστους (cost function) ονομάζεται ο μέσος όρος των τιμών των συναρτήσεων απώλειας. Ιδιαίτερη είναι η σημασία της για την ανεύρεση των βαρών και της πόλωσης του δικτύου. Συγκεκριμένα, υπολογίζεται πρώτα η απώλεια σε κάθε δεδομένο που αφορά τη διαδικασία δοκιμής/πρόβλεψης. Έπειτα, υπολογίζεται ο μέσος όρος των τιμών αυτών, ο οποίος αποτελεί το κόστος. Οι πιο βασικές μορφές εμφάνισης της συνάρτησης κόστους είναι η τετραγωνική συνάρτηση κόστους και η συνάρτηση διασταυρούμενης εντροπίας κόστους.

Συνάρτηση τετραγωνικού κόστους

Η συνάρτηση τετραγωνικού κόστους, η οποία κάποιες φορές ονομάζεται και μέσο τετραγωνικό σφάλμα (mean squared error ή MSE), ορίζεται ως εξής:

$$C(w,b) = \frac{1}{2n}\sum_x \|y(x) - a^l(x)\|^2,$$

όπου w είναι η συλλογή όλων των βαρών του δικτύου, b είναι το σύνολο των πολώσεων, n είναι ο συνολικός αριθμός των δεδομένων εισόδου, α είναι το διάνυσμα των δεδομένων εξόδου του δικτύου όταν το x είναι η είσοδος, y είναι η επιθυμητή έξοδος και το άθροισμα είναι για όλα τα δεδομένα εκπαίδευσης εισόδου x. Βέβαια, η έξοδος α εξαρτάται από τα x, w και b, αλλά για απλούστευση δεν αναλύουμε αυτή την εξάρτηση. Ο συμβολισμός $\|v\|$ υποδηλώνει τη συνήθη συνάρτηση μήκους για ένα διάνυσμα v. Από τη μορφή του τύπου φαίνεται ότι το $C(w,b)$ είναι μη αρνητικό, αφού όλοι οι όροι στο άθροισμα είναι μη αρνητικοί. Επίσης, το κόστος $C(w,b)$ γίνεται μικρό, δηλαδή περίπου ίσο με το μηδέν, ακριβώς όταν το y(x) είναι περίπου ίσο με την έξοδο α, για κάθε δεδομένο εκπαίδευσης εισόδου x. Συνεπώς, ο αλγόριθμος εκπαίδευσης είναι αποδοτικός όταν είναι σε θέση να βρίσκει βάρη και πολώσεις τέτοια ώστε $C(w,b) \approx 0$. Αντίθετα, ο αλγόριθμος εκπαίδευσης δεν είναι και τόσο αποδοτικός όταν το $C(w,b)$ είναι μεγάλο. Αυτό θα σήμαινε ότι το y(x) δεν είναι κοντά στην έξοδο α για ένα μεγάλο αριθμό εισόδων. Άρα, ο στόχος του αλγόριθμου εκπαίδευσης είναι η ελαχιστοποίηση του κόστους $C(w,b)$. Με άλλα λόγια, πρέπει να βρεθεί ένα σετ βαρών και πολώσεων το οποίο κάνει το κόστος όσο μικρότερο γίνεται. Αυτό γίνεται χρησιμοποιώντας αλγορίθμους σαν την κατάβαση κλίσης, για τους οποίους θα γίνει ανάλυση σε επόμενη ενότητα.

Συνάρτηση κόστους διασταυρούμενης εντροπίας

Κατά τη διάρκεια της εκπαίδευσης ενός δικτύου, κάποιες φορές εμφανίζεται το πρόβλημα της επιβράδυνσης της μάθησης. Το πρόβλημα αυτό επιλύεται αντικαθιστώντας τη συνάρτηση τετραγωνικού κόστους με τη συνάρτηση κόστους διασταυρούμενης εντροπίας, η οποία ορίζεται ως εξής:

$$C = -\frac{1}{n}\sum_x [y ln a + (1-y)\ln(1-a)],$$

όπου n είναι ο συνολικός αριθμός των δεδομένων εκπαίδευσης, το άθροισμα είναι για όλα τα δεδομένα εκπαίδευσης εισόδου x και y είναι η αντίστοιχη επιθυμητή έξοδος. Δύο χαρακτηριστικά κάνουν τη διασταυρούμενη εντροπία κατάλληλη για συνάρτηση κόστους. Πρώτον, από τη μορφή του τύπου φαίνεται ότι το C είναι μη αρνητικό, αφού κάθε όρος στο άθροισμα είναι αρνητικός, καθώς και οι δύο λογάριθμοι είναι αριθμών που κυμαίνονται από 0 έως 1, και υπάρχει το μείον μπροστά από το άθροισμα. Δεύτερον, αν η έξοδος ενός νευρώνα είναι κοντά στην επιθυμητή έξοδο για όλα τα δεδομένα εκπαίδευσης εισόδου x, τότε η διασταυρούμενη εντροπία θα είναι κοντά στο μηδέν. Για παράδειγμα, έστω ότι $y = 0$ και $a \approx 0$ για κάποια είσοδο x. Αυτή είναι μία περίπτωση όπου ο νευρώνας είναι αποδοτικός στην είσοδο αυτή. Τότε ο πρώτος όρος του αθροίσματος μηδενίζεται και ο δεύτερος όρος γίνεται

$-\ln(1-a) \approx 0$. Όμοια και αν $y = 1$ και $a \approx 1$. Επομένως, η συνεισφορά στο κόστος θα είναι χαμηλή αν η έξοδος είναι κοντά στην επιθυμητή έξοδο.

2.4 Διασταυρούμενη Εντροπία

Η διασταυρούμενη εντροπία (cross-entropy) αποτελεί ένα μέτρο της διαφοράς μεταξύ δύο κατανομών πιθανότητας για μία δοθείσα τυχαία μεταβλητή ή σύνολο γεγονότων. Η διασταυρούμενη εντροπία είναι ο μέσος αριθμός των bits που χρειάζονται για την κωδικοποίηση δεδομένων προερχόμενων από κάποια πηγή με κατανομή P όταν χρησιμοποιείται ένα μοντέλο Q. Διαισθητικά, αυτό προκύπτει αν θεωρήσουμε μία κατανομή πιθανότητας – στόχο P και μία προσέγγιση της κατανομής – στόχου Q. Τότε η διασταυρούμενη εντροπία της Q από την P είναι ο αριθμός των πρόσθετων bits για την αναπαράσταση ενός γεγονότος με χρήση της Q, αντί της P. Η διασταυρούμενη εντροπία μεταξύ δύο κατανομών πιθανότητας, όπως η Q και η P, γράφεται $H(P, Q)$ και μπορεί να υπολογιστεί με χρήση των πιθανοτήτων των γεγονότων από τις κατανομές P και Q ως εξής:

$$H(P, Q) = -\sum_{x \in X} P(x) * \log(Q(x)),$$

όπου P(x) είναι η πιθανότητα του γεγονότος x στην P και Q(x) είναι η πιθανότητα του γεγονότος x στην Q.

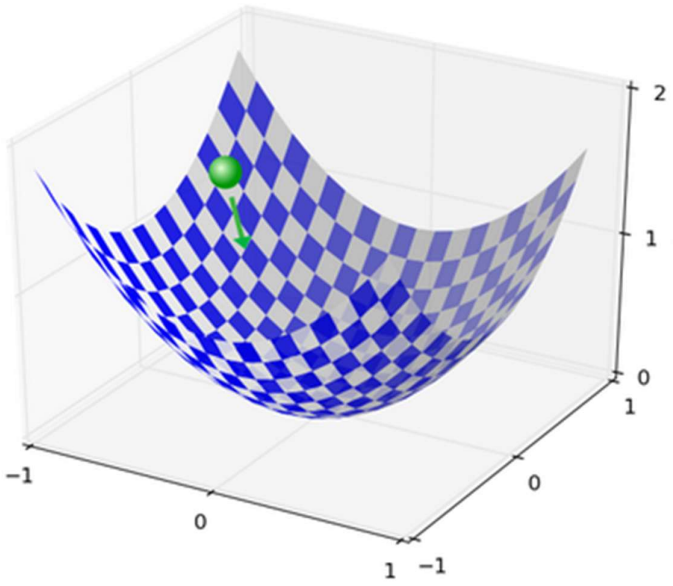
Κατά τη βελτιστοποίηση μοντέλων ταξινόμησης, όπως τα τεχνητά νευρωνικά δίκτυα, η διασταυρούμενη εντροπία μπορεί να χρησιμοποιηθεί ως μία συνάρτηση απώλειας, καθώς και για να εκφραστούν και άλλες σημαντικές συναρτήσεις, όπως η συνάρτηση κόστους.

2.5 Αλγόριθμοι Βελτιστοποίησης

Ένας αλγόριθμος βελτιστοποίησης βρίσκει την τιμή των παραμέτρων (βάρη και ρυθμός εκμάθησης) η οποία ελαχιστοποιεί το σφάλμα κατά την αντιστοίχιση (mapping) των εισόδων στις εξόδους. Οι αλγόριθμοι βελτιστοποίησης, ή αλλιώς βελτιστοποιητές, επηρεάζουν την ακρίβεια του δικτύου σε μεγάλο βαθμό. Ακόμη, επηρεάζουν την ταχύτητα της εκπαίδευσης του δικτύου. Στην ενότητα αυτή θα μελετηθούν βελτιστοποιητές οι οποίοι χρησιμοποιούνται για την υλοποίηση μοντέλων βαθιάς μάθησης, μεταξύ των οποίων και ο Adam, ο οποίος χρησιμοποιήθηκε για τον σκοπό της παρούσας εργασίας στον κώδικα που υλοποιείται παρακάτω.

Κατάβαση κλίσης

Καταρχάς, ιδιαίτερα σημαντική είναι η αποσαφήνιση της έννοιας της κλίσης (gradient). Διαισθητικά, ας θεωρήσουμε την εικόνα μιας κοιλάδας από την κορυφή της οποίας αφήνεται μία μπάλα. Η μπάλα θα κυλήσει κατά μήκος της πιο απότομης κατεύθυνσης της κοιλάδας εωσότου φτάσει στον πυθμένα της. Η κλίση θέτει την μπάλα στην πιο απότομη κατεύθυνση για να φτάσει το τοπικό ελάχιστο, δηλαδή τον πυθμένα της κοιλάδας.



Εικόνα 11: Αναπαράσταση της κοιλάδας και της μπάλας η οποία αφήνεται από την κορυφή της.[11]

Ο αλγόριθμος της κατάβασης κλίσης (gradient descent) χρησιμοποιεί απειροστικό λογισμό για την τροποποίηση των τιμών των παραμέτρων και την επίτευξη του τοπικού ελαχίστου. Ξεκινά με κάποιους συντελεστές, βλέπει το κόστος τους και ψάχνει για κάποια τιμή κόστους μικρότερη από την τρέχουσα. Στη συνέχεια, κινείται προς το χαμηλότερο βάρος και ενημερώνει την τιμή των συντελεστών. Η διαδικασία αυτή επαναλαμβάνεται μέχρι να επιτευχθεί το τοπικό ελάχιστο. Το τοπικό ελάχιστο είναι ένα σημείο πέρα από το οποίο δεν μπορεί να προχωρήσει.

Μειονέκτημα της κατάβασης κλίσης αποτελεί η ακριβή διαδικασία υπολογισμού των κλίσεων αν το σύνολο δεδομένων είναι πολύ μεγάλο. Ακόμη, αν και είναι κατάλληλη για κυρτές συναρτήσεις, δε συμβαίνει το ίδιο και για τις μη κυρτές.

Στοχαστική κατάβαση κλίσης

Ο αλγόριθμος της στοχαστικής κατάβασης κλίσης (stochastic gradient descent) αποφεύγει το πρόβλημα που αντιμετωπίζει η κατάβαση κλίσης με τα μεγάλα σύνολα δεδομένων. Με τον όρο «στοχαστική» εκφράζεται η τυχαιότητα στην οποία είναι βασισμένος ο αλγόριθμος αυτός. Στη στοχαστική κατάβαση κλίσης επιλέγονται τυχαία παρτίδες των δεδομένων, αντί να λαμβάνεται ολόκληρο το σύνολο δεδομένων για κάθε επανάληψη. Αυτό σημαίνει ότι λαμβάνονται μόνο κάποια από τα δείγματα του συνόλου δεδομένων. Ο αλγόριθμος ξεκινά με την επιλογή των αρχικών παραμέτρων των βαρών και του ρυθμού εκμάθησης. Έπειτα, κάνει τυχαίες ανακατατάξεις των δεδομένων σε κάθε επανάληψη για να φτάσει κατά προσέγγιση σε κάποιο ελάχιστο.

Σημαντικό είναι να αναφερθεί ότι αφού δε χρησιμοποιείται ολόκληρο το σύνολο δεδομένων, αλλά παρτίδες αυτού για κάθε επανάληψη, το μονοπάτι το οποίο πήρε ο αλγόριθμος είναι γεμάτο θόρυβο συγκριτικά με την κατάβαση κλίσης. Για το λόγο αυτό, η στοχαστική κατάβαση κλίσης κάνει περισσότερες επαναλήψεις για να φτάσει στο τοπικό ελάχιστο. Εξαιτίας του μεγαλύτερου αριθμού επαναλήψεων, αυξάνεται και ο συνολικός υπολογιστικός

[11] Πηγή: http://neuralnetworksanddeeplearning.com/chap1.html .

χρόνος, αλλά το υπολογιστικό κόστος παραμένει μικρότερο από αυτό της κατάβασης κλίσης. Επομένως, ένα ασφαλές συμπέρασμα που μπορεί να εξαχθεί για την περίπτωση που το σύνολο δεδομένων είναι εξαιρετικά μεγάλο και ο υπολογιστικός χρόνος είναι σημαντικός παράγοντας, είναι ότι είναι προτιμότερη η επιλογή στοχαστικής κατάβασης κλίσης από την κατάβαση κλίσης με παρτίδες.

Adagrad

Ο αλγόριθμος Adagrad (adaptive gradient descent) χρησιμοποιεί διαφορετικούς ρυθμούς εκμάθησης για κάθε επανάληψη. Η αλλαγή του ρυθμού εκμάθησης εξαρτάται από τη διαφορά των παραμέτρων κατά τη διάρκεια της εκπαίδευσης. Όσο περισσότερο αλλάζουν οι παράμετροι, τόσο λιγότερο αλλάζει ο ρυθμός εκμάθησης. Αυτό είναι αρκετά ωφέλιμο, διότι τα πραγματικά σύνολα δεδομένων περιέχουν τόσο αραιά, όσο και πυκνά χαρακτηριστικά. Έτσι, δε φαίνεται σωστό να δοθεί ο ίδιος ρυθμός εκμάθησης για όλα τα χαρακτηριστικά. Το πλεονέκτημα του Adagrad είναι ότι καταργεί την ανάγκη για ορισμό του ρυθμού εκμάθησης χειροκίνητα. Επιπλέον, είναι πιο αξιόπιστος από τους αλγορίθμους κατάβασης κλίσης (gradient descent) και τις παραλλαγές τους και συγκλίνει με μεγαλύτερη ταχύτητα.

Ένα μειονέκτημα του αλγορίθμου Adagrad είναι ότι μειώνει τον ρυθμό εκμάθησης απότομα και με μονοτονία. Απόρροια αυτού είναι ότι σε κάποιο σημείο ο ρυθμός εκμάθησης μπορεί να γίνει υπερβολικά μικρός. Συνεπώς, το δίκτυο καθίσταται ανίκανο να αποκτήσει περισσότερες πληροφορίες και διακινδυνεύεται η ακρίβειά του.

RMS Prop

Ο αλγόριθμος RMS Prop (root mean square) είναι από τους δημοφιλέστερους βελτιστοποιητές. Μπορεί να θεωρηθεί ως μία εξέλιξη του αλγορίθμου Adagrad, καθώς περιορίζει τον μονοτονικά φθίνων ρυθμό εκμάθησης. Ο αλγόριθμος RMS Prop εστιάζει στην επιτάχυνση της διαδικασίας βελτιστοποίησης μειώνοντας τον αριθμό των εκτιμήσεων από τις συναρτήσεις για να φτάσουν το τοπικό ελάχιστο. Με άλλα λόγια, αν υπάρχει μία παράμετρος εξαιτίας της οποίας η συνάρτηση κόστους αμφιταλαντεύεται πολύ, τότε στόχος είναι η «τιμωρία» της ενημέρωσης αυτής της παραμέτρου. Ο αλγόριθμος αυτός συγκλίνει γρήγορα και απαιτεί λιγότερη ρύθμιση από τους αλγορίθμους κατάβασης κλίσης και τις παραλλαγές τους.

Το μειονέκτημα του αλγορίθμου είναι η ανάγκη για ορισμό του ρυθμού εκμάθησης χειροκίνητα, όπου η προτεινόμενη τιμή δε δουλεύει για κάθε εφαρμογή.

Adam

Ο αλγόριθμος Adam (adaptive moment estimation) αποτελεί μία επέκταση της στοχαστικής κατάβασης κλίσης για την ενημέρωση των βαρών του δικτύου κατά την εκπαίδευση. Σε αντίθεση με τη διατήρηση ενός μοναδικού ρυθμού εκμάθησης κατά την εκπαίδευση με στοχαστική κατάβαση κλίσης, ο αλγόριθμος Adam ενημερώνει τον ρυθμό εκμάθησης για κάθε βάρος του δικτύου ξεχωριστά. Ο βελτιστοποιητής Adam συνδυάζει τα πλεονεκτήματα των βελτιστοποιητών Adagrad και RMS Prop. Ο Adam, αντί να προσαρμόζει τους ρυθμούς εκμάθησης με βάση την πρώτη ροπή[12] (μέση τιμή) όπως κάνει ο RMS Prop, χρησιμοποιεί και τη δεύτερη ροπή των κλίσεων, δηλαδή τη μη κεντραρισμένη διακύμανση. Γενικά, ο Adam έχει πολλά πλεονεκτήματα και, συνεπώς, χρησιμοποιείται ευρέως. Ο αλγόριθμος είναι απλός στην εφαρμογή, τρέχει γρηγορότερα, έχει χαμηλές απαιτήσεις όσον αφορά στη μνήμη και απαιτεί λιγότερες παραμέτρους ώστε να ρυθμιστεί από κάθε άλλο αλγόριθμο βελτιστοποίησης. Απότοκο αυτού είναι να προτείνεται ως προεπιλογή για τις περισσότερες εφαρμογές.

[12] Στα μαθηματικά, οι ροπές είναι μεγέθη του σχήματος μιας συνάρτησης. Αν η συνάρτηση είναι κατανομή πιθανότητας, η μηδενική ροπή είναι η συνολική πιθανότητα, δηλαδή 1, η πρώτη ροπή είναι η μέση τιμή, η δεύτερη ροπή είναι η διακύμανση, η Τρίτη ροπή είναι η λοξότητα και η τέταρτη ροπή είναι η κύρτωση της συνάρτησης. Η μαθηματική έννοια συνδέεται άμεσα με την έννοια της ροπής στη φυσική.

Μειονέκτημα του αλγορίθμου αποτελεί η τάση του να εστιάζει στον ταχύτερο υπολογιστικό χρόνο, ενώ αλγόριθμοι όπως η στοχαστική κατάβαση κλίσης εστιάζουν στα data points, δηλαδή τα παραδείγματα του συνόλου δεδομένων.

Κλείνοντας την ενότητα αυτή, παρατηρούμε ότι δεν υπάρχει ένας αλγόριθμος βελτιστοποίησης κατάλληλος για όλα τα δίκτυα. Έτσι, ο αλγόριθμος επιλέγεται ανάλογα με τις απαιτήσεις και τον τύπο των δεδομένων κάθε φορά.

2.6 Εξομάλυνση

Πολλές φορές ένα δίκτυο είναι αποδοτικό για κάποιο σύνολο δεδομένων, αλλά αποτυγχάνει στη γενίκευση η οποία περιλαμβάνει νέες καταστάσεις. Η αληθινή δοκιμασία για ένα δίκτυο είναι η ικανότητα να κάνει προβλέψεις σε καταστάσεις στις οποίες δεν είχε εκτεθεί έως εκείνη τη στιγμή. Ωστόσο, όταν το δίκτυο είναι αρκετά σύνθετο, κάποιες φορές αρχίζει να μαθαίνει τις άσχετες πληροφορίες στο σύνολο δεδομένων. Αυτό σημαίνει ότι το δίκτυο κρατά στη μνήμη το θόρυβο ο οποίος είναι στενά συνδεδεμένος μόνο με το σύνολο δεδομένων εκπαίδευσης. Έτσι, το δίκτυο γίνεται αρκετά ανακριβές, επειδή το μοτίβο που κρατά στη μνήμη δεν αντικατοπτρίζει τη σημαντική πληροφορία που υπάρχει στα δεδομένα. Ένα τέτοιο δίκτυο λέγεται ότι έχει πάθει υπερπροσαρμογή (overfitting) και δεν είναι σε θέση να κάνει ορθή γενίκευση σε νέα δεδομένα. Έχει μάθει εξαιρετικά καλά τα χαρακτηριστικά των δεδομένων εκπαίδευσης, αλλά αν του δοθούν δεδομένα τα οποία αποκλίνουν ελαφρώς από τα δεδομένα που χρησιμοποιήθηκαν κατά τη διάρκεια της εκπαίδευσης, δεν είναι σε θέση να γενικεύσει και να προβλέψει με ακρίβεια την έξοδο. Η υπερπροσαρμογή είναι ένα σημαντικό πρόβλημα στα νευρωνικά δίκτυα. Αυτό ισχύει ιδιαίτερα στα σύγχρονα δίκτυα, τα οποία συχνά έχουν πολύ μεγάλο αριθμό βαρών και πολώσεων. Για να γίνεται, λοιπόν, αποδοτική εκπαίδευση των δικτύων, υπάρχουν τεχνικές για τη μείωση των επιπτώσεων της υπερπροσαρμογής.

Συνήθεις τέτοιες τεχνικές είναι η απλοποίηση του δικτύου, η παύση της διαδικασίας της εκπαίδευσης πριν το δίκτυο αρχίσει να μαθαίνει τις άσχετες πληροφορίες, ή αλλιώς θόρυβο, η προσθήκη περισσότερων δεδομένων στο σύνολο των δεδομένων εκπαίδευσης, η ενίσχυση των δεδομένων εκπαίδευσης με μικρές τροποποιήσεις στα ήδη υπάρχοντα δεδομένα (data augmentation), οι τεχνικές εξομάλυνσης, μεταξύ των οποίων η L1 και η L2, και η Dropout. Στην εργασία αυτή περαιτέρω ανάλυση θα γίνει στις δύο τελευταίες, οι οποίες χρησιμοποιήθηκαν και στην υλοποίηση του προγράμματος παρακάτω για την αυτόματη αναγνώριση του συμβόλου της τελείας σε μαθηματικά κείμενα.

L2 Εξομάλυνση

Η κεντρική ιδέα πίσω από την L2 εξομάλυνση (L2 regularization), η οποία συχνά καλείται και μείωση βάρους, είναι η προσθήκη ενός ακόμα όρου στη συνάρτηση κόστους, ο οποίος ονομάζεται όρος εξομάλυνσης. Ο τύπος της εξομαλυμένης διασταυρούμενης εντροπίας είναι ο εξής:

$$C = -\frac{1}{n}\sum_{xj}\left[y_j ln a_j^L + \left(1-y_j\right)\ln\left(1-a_j^L\right)\right] + \frac{\lambda}{2n}\sum_w w^2.$$

Ο πρώτος όρος είναι η συνήθης έκφραση της διασταυρούμενης εντροπίας. Αλλά προστέθηκε και ο δεύτερος όρος, ο οποίος αποτελεί το άθροισμα των τετραγώνων όλων των βαρών του δικτύου. Το άθροισμα αυτό πολλαπλασιάζεται με $\lambda/2n$, όπου το λ>0 είναι η παράμετρος εξομάλυνσης και το n, όπως και προηγουμένως, είναι το μέγεθος του συνόλου δεδομένων εκπαίδευσης.

Βέβαια, η εξομάλυνση και άλλων συναρτήσεων κόστους είναι δυνατή, όπως της συνάρτησης τετραγωνικού κόστους:

$$C = \frac{1}{2n}\sum_x \|y - a^L\|^2 + \frac{\lambda}{2n}\sum_w w^2.$$

Και στις δύο περιπτώσεις, η εξομαλυμένη συνάρτηση κόστους μπορεί να γραφεί ως εξής:

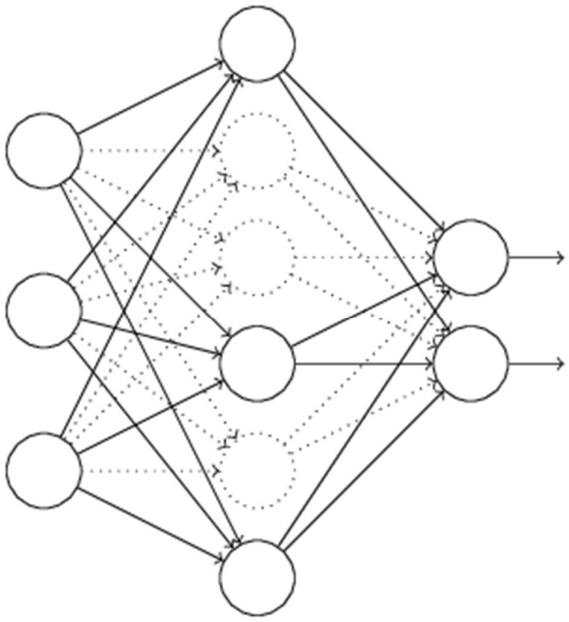
$$C = C_0 + \frac{\lambda}{2n}\sum_w w^2,$$

Όπου το C_0 είναι η αρχική, μη εξομαλυμένη συνάρτηση κόστους.

Διαισθητικά, η εξομάλυνση κάνει το δίκτυο να προτιμά να μάθει μικρά βάρη. Τα μεγάλα βάρη επιτρέπονται μόνο στην περίπτωση που βελτιώνουν σημαντικά το πρώτο κομμάτι της συνάρτησης κόστους. Με άλλα λόγια, η εξομάλυνση αποτελεί έναν τρόπο συμβιβασμού μεταξύ της εύρεσης μικρών βαρών και της ελαχιστοποίησης της αρχικής συνάρτησης κόστους. Η σημασία των δύο στοιχείων του συμβιβασμού αυτού εξαρτάται από την τιμή του λ. Όταν το λ είναι μικρό, είναι προτιμότερη η ελαχιστοποίηση της αρχικής συνάρτησης κόστους, ενώ όταν το λ είναι μεγάλο είναι προτιμότερα τα μικρά βάρη. Έπειτα, λοιπόν, από την εφαρμογή αυτής της μεθόδου, θα είναι δυνατή η μείωση της απόκλισης του δικτύου.

Dropout

Η dropout αποτελεί μία εντελώς διαφορετική τεχνική εξομάλυνσης. Σε αντίθεση με τις L1 και L2 εξομαλύνσεις, η dropout δε βασίζεται στην τροποποίηση της συνάρτησης κόστους. Στη dropout γίνεται τροποποίηση του δικτύου. Όσον αφορά στο πώς ακριβώς λειτουργεί, έστω ένα δεδομένο εκπαίδευσης εισόδου x και η αντίστοιχη επιθυμητή έξοδος y. Κατά κανόνα, η εκπαίδευση θα γινόταν διαδίδοντας προς τα μπροστά στο δίκτυο το x και, έπειτα, κάνοντας οπισθοδιάδοση ώστε να προσδιοριστεί η συνεισφορά στην κλίση. Με την τεχνική dropout, η διαδικασία αυτή τροποποιείται. Αρχικά, γίνεται τυχαία και προσωρινή διαγραφή των μισών κρυφών νευρώνων του δικτύου, με τους νευρώνες εισόδου και εξόδου να μην επηρεάζονται.



Εικόνα 12: Διαγραφή των μισών νευρώνων της τεχνικής dropout.[13]

Στη συνέχεια, γίνεται διάδοση της εισόδου x προς τα μπροστά στο τροποποιημένο δίκτυο και, κατόπιν, οπισθοδιάδοση του αποτελέσματος στο τροποποιημένο δίκτυο. Αφού γίνει αυτό σε ένα mini-batch παραδειγμάτων, γίνεται ενημέρωση των κατάλληλων βαρών και πολώσεων. Η διαδικασία αυτή επαναλαμβάνεται, αποκαθιστώντας πρώτα τους dropout νευρώνες και, ακολούθως, επιλέγοντας ένα νέο τυχαίο υποσύνολο κρυφών νευρώνων για διαγραφή,

[13] Πηγή: http://neuralnetworksanddeeplearning.com/chap3.html

υπολογίζοντας την κλίση για κάποιο άλλο mini-batch και, τέλος, ενημερώνοντας τα βάρη και τις πολώσεις στο δίκτυο.

Με την επανάληψη της διαδικασίας αυτής ξανά και ξανά, το δίκτυο θα μάθει ένα σετ βαρών και πολώσεων. Βέβαια, αυτά τα βάρη και οι πολώσεις θα μαθευτούν υπό συνθήκες στις οποίες οι μισοί κρυφοί νευρώνες είχαν διαγραφτεί. Όταν εκτελείται ολόκληρο το δίκτυο, διπλάσιοι νευρώνες είναι ενεργοί. Για να αντισταθμιστεί αυτό, μειώνονται στο μισό τα βάρη τα οποία εξέρχονται από τους κρυφούς νευρώνες.

Στην τεχνική dropout, με τη διαγραφή διαφορετικών σετ νευρώνων, είναι σαν να εκπαιδεύονται διαφορετικά δίκτυα. Κατά συνέπεια, είναι σαν να βγαίνει ο μέσος όρος των επιδράσεων ενός πολύ μεγάλου αριθμού διαφορετικών δικτύων. Τα διαφορετικά δίκτυα θα πάθουν υπερπροσαρμογή με διαφορετικούς τρόπους και, έτσι, η επίδραση της dropout στο δίκτυο θα είναι να μειώσει την υπερπροσαρμογή.

Κλείνοντας την ενότητα αυτή, ιδιαίτερα σημαντικό είναι να αναφερθεί ότι είναι δυνατός ο συνδυασμός τεχνικών εξομάλυνσης. Για παράδειγμα, στο πρόγραμμα που υλοποιήθηκε για τις ανάγκες της παρούσας εργασίας συνδυάστηκε η L2 εξομάλυνση με τη dropout, ώστε να επιτευχθούν τα καλύτερα δυνατά αποτελέσματα.

ΚΕΦΑΛΑΙΟ 3

ΕΠΑΝΑΛΑΜΒΑΝΟΜΕΝΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ (ΕΝΔ)

Η σκέψη των ανθρώπων έχει συνοχή. Αντί, δηλαδή, να τα ξεχνούν όλα και να ξεκινούν από το μηδέν κάθε φορά, βασίζονται σε προηγούμενες σκέψεις και γνώσεις τους για να σκεφτούν κάτι καινούργιο.

Τα νευρωνικά δίκτυα πρόσθιας τροφοδότησης (feedforward networks), ο πρώτος και πιο απλός τύπος τεχνητού νευρωνικού δικτύου που αναπτύχθηκε, δεν μπορούν να το κάνουν αυτό. Στα δίκτυα πρόσθιας τροφοδότησης υπάρχει μία μοναδική είσοδος η οποία καθορίζει πλήρως τις ενεργοποιήσεις όλων των νευρώνων στα υπόλοιπα επίπεδα. Είναι μία πολύ στατική εικόνα, αφού όλα στο δίκτυο είναι φιξαρισμένα. Ας υποθέσουμε ότι επιτρέπουμε στα στοιχεία του δικτύου να αλλάζουν με δυναμικό τρόπο. Για παράδειγμα, η συμπεριφορά των κρυφών νευρώνων μπορεί να μην καθορίζεται μόνο από τις ενεργοποιήσεις στα προηγούμενα κρυφά επίπεδα, αλλά και από τις ενεργοποιήσεις από προηγούμενες φορές. Πράγματι, η ενεργοποίηση ενός νευρώνα μπορεί να καθοριστεί εν μέρει από τη δική του ενεργοποίηση σε προηγούμενη φορά. Αυτό δε συμβαίνει σε ένα δίκτυο πρόσθιας τροφοδότησης. Ή ίσως οι ενεργοποιήσεις των κρυφών νευρώνων και των νευρώνων εξόδου δε θα καθορίζονται μόνο από την τρέχουσα είσοδο στο δίκτυο, αλλά και από προηγούμενες εισόδους.

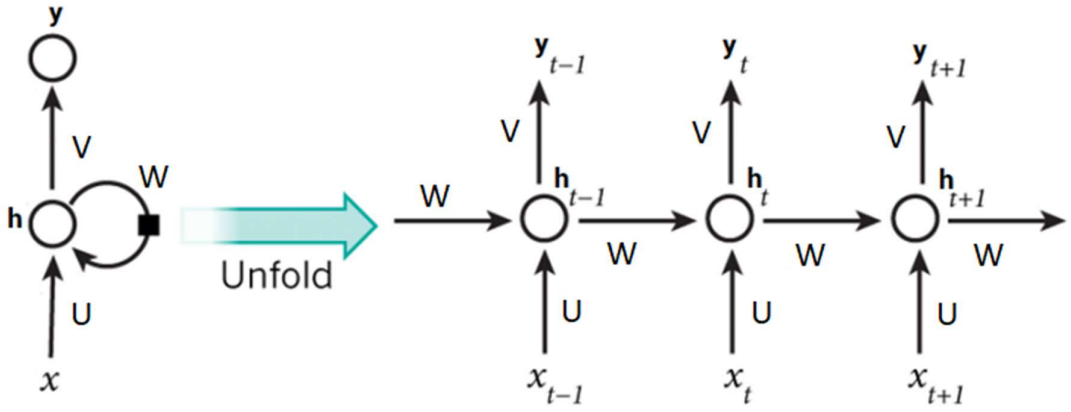
Τα νευρωνικά δίκτυα με αυτή τη συμπεριφορά ονομάζονται επαναλαμβανόμενα νευρωνικά δίκτυα (Recurrent Neural Networks ή RNNs). Η κύρια ιδέα είναι ότι τα επαναλαμβανόμενα νευρωνικά δίκτυα είναι νευρωνικά δίκτυα στα οποία υπάρχει κάποια έννοια δυναμικής αλλαγής με την πάροδο του χρόνου. Είναι ιδιαίτερα χρήσιμα στην ανάλυση δεδομένων ή διαδικασιών που αλλάζουν με την πάροδο του χρόνου. Τέτοια δεδομένα και διαδικασίες προκύπτουν σε προβλήματα όπως ο λόγος ή η φυσική γλώσσα, για παράδειγμα.

Ένα από τα πρώτα παραδείγματα δικτύου που ενσωματώνει επαναλήψεις είναι το δίκτυο Hopfield, το οποίο εισήχθη από τον Αμερικανό επιστήμονα John Hopfield το 1982. Όμοια με τους νευρώνες McCulloch – Pitts, οι νευρώνες Hopfield είναι δυαδικές μονάδες κατωφλίου, αλλά με επαναλαμβανόμενες, αντί για προς τα εμπρός (feedforward) συνδέσεις. Οι μονάδες αυτές είναι αμφίδρομα συνδεδεμένες μεταξύ τους. Μολονότι τα δίκτυα Hopfield ήταν καινοτόμα, το πρώτο επιτυχές παράδειγμα επαναλαμβανόμενου δικτύου εκπαιδευμένου με οπισθοδιάδοση (backpropagation) είναι το δίκτυο Elman, το οποίο εισήχθη από τον Αμερικανό ψυχογλωσσολόγο και καθηγητή γνωσιακής επιστήμης Jeffrey Elman το 1990. Ο Elman πραγματοποίησε πολλά πειράματα, αποδεικνύοντας ότι το δίκτυό του ήταν σε θέση να λύσει διάφορα προβλήματα με διαδοχική δομή. Κάποια από αυτά είναι μία έκδοση που σχετίζεται με το χρόνο του προβλήματος XOR, το να μάθει τη δομή (για παράδειγμα τη διαδοχική σειρά των φωνηέντων και των συμφώνων) σε ακολουθίες γραμμάτων, καθώς και να μάθει, επίσης, σύνθετες λεξικολογικές κλάσεις, όπως η σειρά των λέξεων σε μικρές προτάσεις.

Όσον αφορά στο πώς τα επαναλαμβανόμενα νευρωνικά δίκτυα λειτουργούν, πολλά από αυτά που χρησιμοποιούνται στα δίκτυα πρόσθιας τροφοδότησης μπορούν να χρησιμοποιηθούν και στα επαναλαμβανόμενα νευρωνικά δίκτυα. Συγκεκριμένα, μπορούμε να εκπαιδεύσουμε τα επαναλαμβανόμενα νευρωνικά δίκτυα χρησιμοποιώντας απλές τροποποιήσεις στην κλίση κατάβασης (gradient descent) και την οπισθοδιάδοση (backpropagation). Πολλές άλλες ιδέες που χρησιμοποιούνται σε δίκτυα πρόσθιας τροφοδότησης, από τεχνικές εξομάλυνσης σε συνελίξεις έως τις συναρτήσεις ενεργοποίησης και κόστους, είναι επίσης χρήσιμες στα επαναλαμβανόμενα νευρωνικά δίκτυα.

Απλούστερα, τα επαναλαμβανόμενα νευρωνικά δίκτυα είναι δίκτυα με βρόχους που χρησιμοποιούν διαδοχικές πληροφορίες. Ένα επαναλαμβανόμενο νευρωνικό δίκτυο μπορεί να θεωρηθεί ως πολλαπλά αντίγραφα του ίδιου δικτύου, καθένα από τα οποία περνά ένα

μήνυμα σε ένα επόμενο. Ας ξεδιπλώσουμε (unfold) ένα απλό επαναλαμβανόμενο νευρωνικό δίκτυο σε ένα υπολογιστικό γράφημα (computational graph) :



Εικόνα 13: Ένα ξεδιπλωμένο RNN.[14]

Όπου:

- x_t: η είσοδος τη χρονική στιγμή t.
- h_t: η κρυφή κατάσταση τη χρονική στιγμή t. Η ονομασία αυτή προκύπτει από το γεγονός ότι οι τιμές εκπαίδευσης, δηλαδή η είσοδος την τρέχουσα χρονική στιγμή και η προηγούμενη κρυφή κατάσταση, δε φαίνονται. Υπολογίζεται από τον τύπο $h_t = f(U * x_t + W * s_{t-1})$.
- y_t: η έξοδος τη χρονική στιγμή t η οποία αναπαρίσταται ως πιθανότητα στο εύρος τιμών [0,1], $y_t = softmax(V * h_t)$.

Η επαναλαμβανόμενη μορφή του δείχνει ότι τα επαναλαμβανόμενα νευρωνικά δίκτυα είναι στενά συνδεδεμένα με ακολουθίες και λίστες. Το ξεδίπλωμα αυτού του γραφήματος έχει ως αποτέλεσμα το «μοίρασμα» των παραμέτρων U, V, W σε όλη την έκταση μιας δομής βαθέος δικτύου (deep network), δηλαδή εκτελείται η ίδια ενέργεια κάθε φορά, αλλά με διαφορετικές εισόδους.

Μερικά παραδείγματα σημαντικών μοτίβων σχεδιασμού επαναλαμβανόμενων νευρωνικών δικτύων είναι τα εξής:

- επαναλαμβανόμενα νευρωνικά δίκτυα που παράγουν μία έξοδο σε κάθε βήμα και έχουν επαναλαμβανόμενες συνδέσεις μεταξύ των κρυφών μονάδων.
- επαναλαμβανόμενα νευρωνικά δίκτυα που παράγουν μία έξοδο σε κάθε βήμα και έχουν επαναλαμβανόμενες συνδέσεις μόνο μεταξύ της εξόδου σε μία χρονική στιγμή και των κρυφών μονάδων την επόμενη χρονική στιγμή.
- επαναλαμβανόμενα νευρωνικά δίκτυα με επαναλαμβανόμενες συνδέσεις μεταξύ των κρυφών μονάδων, οι οποίες διαβάζουν μία ολόκληρη ακολουθία και έπειτα παράγουν μία μοναδική έξοδο.

3.1 Μαθηματική Περιγραφή των ΕΝΔ

Όπως και στα δίκτυα πρόσθιας τροφοδότησης, τα βασικά δομικά στοιχεία ενός επαναλαμβανόμενου νευρωνικού δικτύου είναι νευρώνες που συνδέονται μεταξύ τους με συναπτικούς συνδέσμους (συνδέσεις) των οποίων η συναπτική ισχύς κωδικοποιείται από ένα βάρος. Οι νευρώνες αυτοί διακρίνονται σε νευρώνες εισόδου, εσωτερικούς νευρώνες (ή αλλιώς κρυφούς) και νευρώνες εξόδου. Σε μια δεδομένη στιγμή, ένας νευρώνας έχει μία

[14] Πηγή: https://www.analyticsvidhya.com/blog/2017/12/introduction-to-recurrent-neural-networks/ .

ενεργοποίηση. Οι ενεργοποιήσεις των νευρώνων εισόδου συμβολίζονται με $u(n)$, των εσωτερικών νευρώνων με $x(n)$ και των νευρώνων εξόδου με $y(n)$. Ενίοτε, λόγω της στενής εννοιολογικής συγγένειας, αγνοούμε το διαχωρισμό των νευρώνων σε εισόδου/εσωτερικούς/εξόδου και χρησιμοποιούμε $x(n)$ για τις ενεργοποιήσεις και των τριών κατηγοριών.

Υπάρχουν πολλοί τύποι μοντέλων επαναλαμβανόμενων νευρωνικών δικτύων. Τα μοντέλα διακριτού χρόνου απεικονίζονται μαθηματικά ως χάρτες που επαναλαμβάνονται σε διακριτά χρονικά βήματα $n = 1, 2, 3, \dots$. Από την άλλη, τα μοντέλα συνεχούς χρόνου ορίζονται μέσω διαφορικών εξισώσεων των οποίων οι λύσεις ορίζονται σε ένα συνεχή χρόνο t. Στην παρούσα εργασία, θα ασχοληθούμε με μοντέλα διακριτού χρόνου.

Το μοντέλο μας αποτελείται από K νευρώνες εισόδου με διάνυσμα ενεργοποίησης

$$u(n) = \big(u_1(n), \dots, u_K(n)\big)^t,$$

από N εσωτερικούς νευρώνες με διάνυσμα ενεργοποίησης

$$x(n) = \big(x_1(n), \dots, x_N(n)\big)^t,$$

και από L νευρώνες εξόδου με διάνυσμα ενεργοποίησης

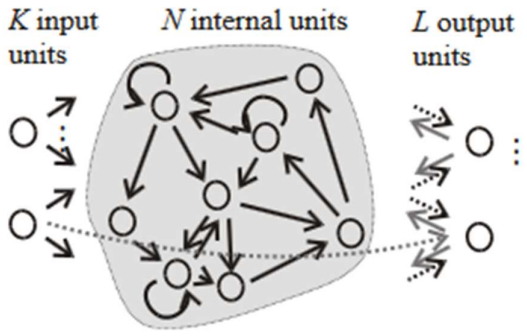
$$y(n) = \big(y_1(n), \dots, y_L(n)\big)^t,$$

όπου ο εκθέτης t δηλώνει αναστροφή. Τα βάρη των συνδέσεων εισόδου/εσωτερικών/εξόδου συγκεντρώνονται σε αντίστοιχους πίνακες βαρών $N \times K / N \times N / L \times (K + N)$ διαστάσεων:

$$W^{in} = (w_{ij}^{in}), W = (w_{ij}), W^{out} = (w_{ij}^{out}).$$

Οι νευρώνες εξόδου μπορούν προαιρετικά να κατευθυνθούν προς τα πίσω στους εσωτερικούς νευρώνες με συνδέσεις των οποίων τα βάρη συγκεντρώνονται σε έναν πίνακα βαρών $N \times L$ διαστάσεων:

$$W^{back} = (w_{ij}^{back}).$$



Εικόνα 14: Η βασική αρχιτεκτονική του δικτύου. Τα γκρι βέλη παριστάνουν τις προαιρετικές συνδέσεις, ενώ τα βέλη με τις κουκκίδες παριστάνουν συνδέσεις εκπαιδευμένες με προσέγγιση δικτύου κατάστασης ηχούς.[15]

Η μηδενική τιμή βάρους σημαίνει ότι δεν υπάρχει σύνδεση. Ακόμη, αξίζει να επισημανθεί ότι οι νευρώνες εξόδου είναι πιθανό να έχουν συνδέσεις όχι μόνο από εσωτερικούς νευρώνες, αλλά, συχνά, και από νευρώνες εισόδου και, σπανιότερα, από νευρώνες εξόδου.

[15] Πηγή: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.4095&rep=rep1&type=pdf .

Η ενημέρωση της ενεργοποίησης των εσωτερικών νευρώνων γίνεται με τον παρακάτω τύπο:

$$(M1) \quad x(n+1) = f(W^{in}u(n+1) + Wx(n) + W^{back}y(n)),$$

όπου $u(n+1)$ είναι η εξωτερικά δοθείσα είσοδος και το f παριστάνει την στοιχείο προς στοιχείο εφαρμογή της συνάρτησης μεταφοράς f του ατομικού νευρώνα (επίσης γνωστή ως συνάρτηση ενεργοποίησης). Πολλές φορές χρησιμοποιείται η υπερβολική εφαπτομένη που είδαμε σε προηγούμενη ενότητα $f = tanh$, αλλά σε περιπτώσεις γραμμικών δικτύων πιθανόν να έχουμε $f = 1$. Η έξοδος υπολογίζεται σύμφωνα με τον τύπο:

$$(M2) \quad y(n+1) = f^{out}(W^{out}(u(n+1), x(n+1), y(n)),$$

όπου $(u(n+1), x(n+1), y(n))$ παριστάνει το διάνυσμα που προκύπτει από τη συνένωση των διανυσμάτων των ενεργοποιήσεων των νευρώνων εισόδου, των εσωτερικών νευρώνων και των νευρώνων εξόδου. Σε αντιστοιχία με αυτό που αναφέρθηκε παραπάνω, $f^{out} = tanh$ ή $f^{out} = 1$ (όπου οι νευρώνες εξόδου είναι γραμμικοί).

3.2 Τεχνικές Εκπαίδευσης των ΕΝΔ

Κατά την τελευταία δεκαετία, έχουν ερευνηθεί αρκετές μέθοδοι εποπτευόμενης εκπαίδευσης για τα επαναλαμβανόμενα νευρωνικά δίκτυα. Η πιο διαδεδομένη είναι η οπισθοδιάδοση μέσα στο χρόνο (backpropagation through time ή BPTT), η πιο σαφής από μαθηματική άποψη είναι η επαναλαμβανόμενη μάθηση πραγματικού χρόνου (real-time recurrent learning ή RTRL) και, κατά πολλούς, αυτή που δίνει τα καλύτερα αποτελέσματα είναι το παρατεταμένο φιλτράρισμα Kalman (extended Kalman filtering ή EKF).

Οπισθοδιάδοση (Backpropagation): Η BPTT είναι μία τροποποίηση της μεθόδου οπισθοδιάδοσης που χρησιμοποιείται για την εκπαίδευση των δικτύων πρόσθιας τροφοδότησης. Η βασική ιδέα είναι απλή. Η διαδικασία εκπαίδευσης ξεκινά με τυχαίες τιμές των βαρών του δικτύου. Αν δεν εξαχθεί το επιθυμητό αποτέλεσμα, το δίκτυο συγκρίνει το αποτέλεσμα που έβγαλε με το επιθυμητό και υπολογίζει το σφάλμα για κάθε νευρώνα του επιπέδου εξόδου. Το σφάλμα αυτό μεταφέρεται πίσω σε όλους τους νευρώνες με μια διαδικασία κλειστής επανάληψης. Υπολογίζεται η κλίση (gradient) του σφάλματος σε σχέση με όλα τα βάρη του δικτύου, αντί για το κάθε βάρος ξεχωριστά, γεγονός που καθιστά δυνατή τη χρήση gradient μεθόδων για την εκπαίδευση πολυεπίπεδων δικτύων, διορθώνοντας τα βάρη ώστε να ελαχιστοποιηθεί το σφάλμα. Συνήθως χρησιμοποιείται κατάβαση κλίσης (gradient descent) ή παραλλαγές, όπως στοχαστική κατάβαση κλίσης (stochastic gradient descent). Τα βήματα της μεθόδου είναι τα εξής:

- Θεωρούμε ένα πολυεπίπεδο perceptron (MLP) με k κρυφά επίπεδα. Συνολικά μαζί με το επίπεδο εισόδου και το επίπεδο εξόδου έχουμε k+2 επίπεδα, τα οποία απαριθμούμε από 0 έως k+1. Οι κόμβοι εισόδου είναι K, οι κόμβοι εξόδου είναι L και οι κόμβοι του κρυφού επιπέδου m είναι N^m. Το βάρος του j-οστού κόμβου στο επίπεδο m και του i-οστού κόμβου στο επίπεδο m+1 συμβολίζεται με w_{ij}^m. Η ενεργοποίηση του i-οστού κόμβου στο επίπεδο m συμβολίζεται με x_i^m (για m=0 είναι τιμή εισόδου και για m=k+1 είναι τιμή εξόδου).

Τα δεδομένα εκπαίδευσης για ένα δίκτυο πρόσθιας τροφοδότησης αποτελούνται από T ζεύγη δεδομένων εισόδου/εξόδου (με μορφή διανύσματος):

$$u(n) = \left(x_1^0(n), \dots, x_K^0(n)\right)^t, \; d(n) = \left(d_1^{k+1}(n), \dots, d_L^{k+1}(n)\right)^t,$$

όπου n συμβολίζει ένα δείγμα των δεδομένων και όχι το χρόνο. Η ενεργοποίηση των κόμβων που δεν ανήκουν στο επίπεδο εισόδου υπολογίζεται από τη σχέση:

$$x_i^{m+1}(n) = f\left(\sum_{j=1,\dots,N^m} w_{ij}^m \, x_j(n)\right).$$

Συνήθως υπάρχουν και όροι για την πόλωση τους οποίους εδώ παραλείπουμε. Εκθέτοντας το δίκτυο στα δεδομένα εισόδου u(t), χρησιμοποιεί τον παραπάνω τύπο για να υπολογίσει τις ενεργοποιήσεις των κόμβων στα επόμενα κρυφά επίπεδα, ώσπου να δώσει ένα αποτέλεσμα:

$y(n) = (x_1^{k+1}(n), \dots, x_L^{k+1}(n))'.$

Ο στόχος της εκπαίδευσης είναι η εύρεση ενός σετ βαρών με τα οποία να ελαχιστοποιείται το τετραγωνικό σφάλμα:

$E = \sum_{n=1,\dots,T} \|d(n) - y(n)\|^2 = \sum_{n=1,\dots,T} E(n).$

Αυτό επιτυγχάνεται αλλάζοντας σταδιακά τα βάρη προς την κατεύθυνση της κλίσης του σφάλματος σε σχέση με τα βάρη χρησιμοποιώντας έναν (μικρό) ρυθμό εκπαίδευσης γ:

$\frac{\partial E}{\partial w_{ij}^m} = \sum_{t=1,\dots,T} \frac{\partial E(n)}{\partial w_{ij}^m},$

$new\ w_{ij}^m = w_{ij}^m - \gamma \frac{\partial E}{\partial w_{ij}^m}.$

Αυτός είναι ο τύπος που χρησιμοποιείται στην εκπαίδευση με παρτίδες (batch learning), όπου τα νέα βάρη υπολογίζονται αφού δοθούν στο δίκτυο όλα τα δείγματα της εκπαίδευσης. Ένα τέτοιο πέρασμα από όλα τα δείγματα ονομάζεται εποχή (epoch). Πριν την πρώτη εποχή, τα βάρη αρχικοποιούνται, συνήθως σε μικρές τυχαίες τιμές. (Σε αυτό το σημείο είναι χρήσιμο να υπενθυμίσουμε ότι, γενικά, ο αριθμός των εποχών αντιστοιχεί στο συνολικό αριθμό των επαναλήψεων που θα πραγματοποιήσει το δίκτυο ώστε να εκπαιδευτεί και ο αριθμός των παρτίδων ορίζει τον εσωτερικό αριθμό των επαναλήψεων που πραγματοποιούνται ανά εποχή.) Μία παραλλαγή είναι η σταδιακή μάθηση (incremental learning), όπου τα βάρη αλλάζουν σε κάθε βήμα αφού δοθούν στο σύστημα τα ατομικά δείγματα εκπαίδευσης:

$new\ w_{ij}^m = w_{ij}^m - \gamma \frac{\partial E(n)}{\partial w_{ij}^m}.$

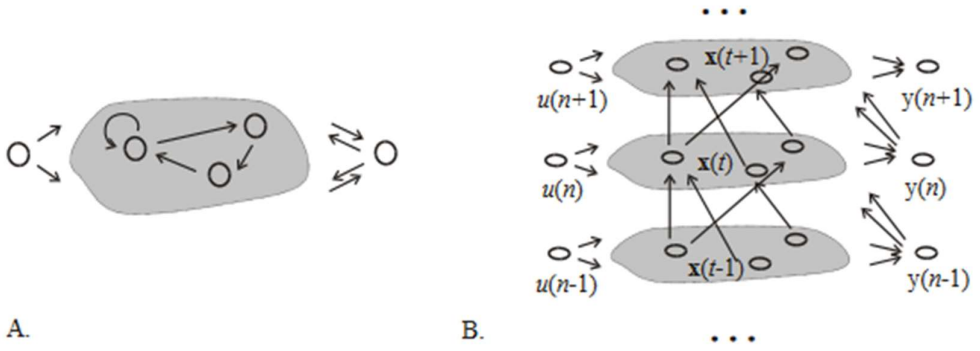
Ο αλγόριθμος οπισθοδιάδοσης επαναλαμβάνει την εφαρμογή των παραπάνω τύπων για κάθε εποχή έως ότου το σφάλμα γίνει μικρότερο από ένα προκαθορισμένο κατώφλι ή η αλλαγή στο σφάλμα γίνει μικρότερη από ένα άλλο προκαθορισμένο κατώφλι ή ο αριθμός των εποχών υπερβεί έναν προκαθορισμένο μέγιστο αριθμό εποχών. Πολλές (της τάξης των χιλιάδων σε μη-τετριμμένες περιπτώσεις) τέτοιες εποχές είναι πιθανό να χρειαστούν μέχρι να επιτευχθεί ένα επαρκώς μικρό σφάλμα.

Μία εποχή απαιτεί O(TM) πολλαπλασιασμούς και προσθέσεις, όπου M είναι ο συνολικός αριθμός συνδέσεων του δικτύου.

Η βασική προσέγγιση κατάβασης κλίσης (gradient descent) (και η υλοποίηση με τον αλγόριθμο οπισθοδιάδοσης) είναι γνωστή για την αργή της σύγκλιση, αφού ο ρυθμός εκμάθησης γ πρέπει να επιλέγεται κατά κανόνα μικρός για να αποφευχθεί η αστάθεια. Ένας τρόπος για να επιτευχθεί πιο γρήγορη σύγκλιση είναι η χρήση δεύτερου βαθμού τεχνικών κατάβασης κλίσης, οι οποίες αξιοποιούν την καμπυλότητα της κλίσης, αλλά έχουν εποχική πολυπλοκότητα $O(TM^2)$. Όπως όλες οι τεχνικές κατάβασης κλίσης, η οπισθοδιάδοση βρίσκει μόνο ένα τοπικό ελάχιστο του σφάλματος. Αυτό το πρόβλημα μπορεί να αντιμετωπιστεί, για παράδειγμα, με την προσθήκη θορύβου κατά τη διάρκεια της εκπαίδευσης ή με την επανάληψη όλης της εκπαιδευτικής διαδικασίας με διαφορετικά αρχικά βάρη ή με τη χρήση συγκεκριμένων πληροφοριών ώστε να ξεκινήσει η διαδικασία από ένα πιθανά κατάλληλο σετ βαρών. Ένα άλλο πρόβλημα είναι η επιλογή της κατάλληλης τοπολογίας του δικτύου, δηλαδή του αριθμού και του μεγέθους των κρυφών επιπέδων. Πάλι μπορούν να χρησιμοποιηθούν πρωτύτερες πληροφορίες, έλεγχος στο σύστημα ή διαίσθηση.

Συνολικά, αρκετή τεχνογνωσία και εμπειρία χρειάζονται για καλά αποτελέσματα σε μη-τετριμμένες περιπτώσεις παρόλο που η βασική οπισθοδιάδοση είναι εύκολα εφαρμόσιμη.

Backpropagation Through Time: Ο αλγόριθμος οπισθοδιάδοσης που χρησιμοποιείται στα δίκτυα πρόσθιας τροφοδότησης δεν μπορεί να εφαρμοστεί αυτούσιος στα επαναλαμβανόμενα νευρωνικά δίκτυα, διότι το σφάλμα το οποίο περνά στο δίκτυο προϋποθέτει ότι οι συνδέσεις μεταξύ των νευρώνων δεν είναι κυκλικές. Το πρόβλημα αυτό αντιμετωπίζεται από την BPTT με το να «ξεδιπλώσει» το επαναλαμβανόμενο νευρωνικό δίκτυο, τοποθετώντας πανομοιότυπα αντίγραφα του επαναλαμβανόμενου νευρωνικού δικτύου το ένα μετά το άλλο, και να ανακατευθύνει τις συνδέσεις μέσα στο δίκτυο ώστε να αποκτήσει συνδέσεις μεταξύ των επακόλουθων αντιγράφων. Με αυτό τον τρόπο κατασκευάζει ένα δίκτυο πρόσθιας τροφοδότησης.



Εικόνα 15: Η βασική ιδέα της BPTT. A: Το αρχικό RNN. B: Το feedforward δίκτυο που προκύπτει από το RNN.[16]

Τα βάρη w_{ij}^{in}, w_{ij}, w_{ij}^{out}, w_{ij}^{back} είναι ίδια σε όλα τα αντίγραφα. Τα δεδομένα εκπαίδευσης αποτελούνται τώρα από μία χρονική σειρά εισόδου/εξόδου:

$$u(n) = \left(u_1(n), \dots, u_K(n)\right)', d(n) = (d_1(n), \dots, d_L(n))', n = 1, \dots, T.$$

Το forward pass (δηλαδή η ανάθεση τιμών σε όλες τις μεταβλητές και ο υπολογισμός κάθε ενδιάμεσης τιμής με βάση τη μορφή της δοσμένης συνάρτησης) μίας εποχής συνίσταται από την ενημέρωση του δικτύου, ξεκινώντας από το πρώτο αντίγραφο και προχωρώντας στα επόμενα. Σε κάθε αντίγραφο/φορά n, διαβάζεται η είσοδος u(n), έπειτα υπολογίζεται η εσωτερική κατάσταση x(n) βάσει των u(n), x(n-1) (και του y(n-1) αν υπάρχει μη μηδενικό w_{ij}^{back}) και, τέλος, υπολογίζεται η έξοδος του τρέχοντος αντίγραφου y(n).

Το σφάλμα προς ελαχιστοποίηση είναι πάλι:

$$(B1) \quad E = \sum_{n=1,\dots,T} \|d(n) - y(n)\|^2 = \sum_{n=1,\dots,T} E(n),$$

αλλά η σημασία του t άλλαξε από «δείγμα εκπαίδευσης» σε «φορά». Ο αλγόριθμος είναι ο εξής:

Είσοδος: τα βάρη της τρέχουσας χρονικής στιγμής w_{ij} και η χρονική σειρά εκπαίδευσης.

Έξοδος: τα νέα βάρη.

Υπολογιστικά βήματα:

1. Forward pass: όπως περιγράφτηκε παραπάνω.

[16] Πηγή: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.4095&rep=rep1&type=pdf .

2. Υπολογισμός, προχωρώντας προς τα πίσω για n=T, …, 1, για κάθε φορά n και ενεργοποίηση κόμβου $x_i(n)$, $y_j(n)$ τον όρο διάδοσης σφάλματος $\delta_i(n)$

$$(B2)\quad \delta_j(T) = (d_j(T) - y_j(T))\frac{\partial f(u)}{\partial u}\Big|_{u=z_{j(T)}}$$

για τους κόμβους εξόδου του χρονικού επιπέδου T και

$$(B3)\quad \delta_i(T) = \left[\sum_{j=1}^{L} \delta_j(T) w_{ji}^{out}\right]\frac{\partial f(u)}{\partial u}\Big|_{u=z_{i(n)}}$$

για τους εσωτερικούς κόμβους $x_i(T)$ στο χρονικό επίπεδο T και

$$(B4)\quad \delta_j(n) = \left[\Big(d_j(n) - y_j(n)\Big) + \sum_{i=1}^{N} \delta_i(n+1) w_{ij}^{back}\right]\frac{\partial f(u)}{\partial u}\Big|_{u=z_{j(n)}}$$

για τους κόμβους εξόδου των προηγούμενων επιπέδων και

$$(B5)\quad \delta_i(n) = \left[\sum_{j=1}^{N} \delta_j(n+1) w_{ji} + \sum_{j=1}^{L} \delta_j(n) w_{ji}^{out}\right]\frac{\partial f(u)}{\partial u}\Big|_{u=z_{i(n)}}$$

για τους εσωτερικούς κόμβους $x_i(n)$ στις προηγούμενες φορές, όπου $z_i(n)$ είναι η μέγιστη τιμή που μπορεί να πάρει ο αντίστοιχος κόμβος.

3. Προσαρμογή των βαρών σύμφωνα με:

$new\ w_{ij} = w_{ij} + \gamma \sum_{n=1}^{T} \delta_i(n) x_j(n-1)$ (με χρήση της $x_j(n-1) = 0$ για $n = 1$)

$$new\ w_{ij}^{in} = w_{ij}^{in} + \gamma \sum_{n=1}^{T} \delta_i(n) u_j(n)$$

$$new\ w_{ij}^{out} = w_{ij}^{out} + \gamma \times \begin{cases} \sum_{n=1}^{T} \delta_i(n) u_j(n), \alpha\nu\ \tau o\ j\ \alpha\phi o\rho\acute{\alpha}\ \kappa\acute{o}\mu\beta o\ \epsilon\iota\sigma\acute{o}\delta o\upsilon \\ \sum_{n=1}^{T} \delta_i(n) x_j(n-1), \alpha\nu\ \tau o\ j\ \alpha\phi o\rho\acute{\alpha}\ \kappa\rho\upsilon\phi\acute{o}\ \kappa\acute{o}\mu\beta o \end{cases}$$

$new\ w_{ij}^{back} = w_{ij}^{back} + \gamma \sum_{n=1}^{T} \delta_i(n) y_j(n-1)$ (με χρήση της $y_j(n-1) = 0$ για $n = 0$)

Η παρατήρηση που έγινε σχετικά με την αργή σύγκλιση για την οπισθοδιάδοση παραμένει και στην BPTT. Η υπολογιστική πολυπλοκότητα για μία εποχή είναι $O(TN^2)$, όπου N είναι ο αριθμός των εσωτερικών κόμβων και, έτσι, συνήθως απαιτούνται αρκετές χιλιάδες εποχές.

Μία παραλλαγή αυτού του αλγορίθμου είναι με τη χρήση της επιθυμητής εξόδου d(n) στον υπολογισμό των ενεργοποιήσεων στο επίπεδο n+1 κατά το forward pass. Αυτό ονομάζεται teacher forcing. Η διαδικασία αυτή επιταχύνει τη σύγκλιση, ή, μάλιστα, κάποιες φορές κρίνεται απαραίτητη ώστε να επιτευχθεί η σύγκλιση, αλλά ενίοτε μπορεί να εμφανίσει αστάθεια. Δεν υπάρχει κάποιος γενικός κανόνας για τη χρήση της.

Ένα μειονέκτημα της BPTT που εφαρμόζει εκπαίδευση με παρτίδες είναι ότι πρέπει να χρησιμοποιηθεί ολόκληρη η χρονική σειρά των δεδομένων εκπαίδευσης. Το γεγονός αυτό αποκλείει περιπτώσεις όπου απαιτείται online προσαρμογή. Λύση αυτού του ζητήματος αποτελεί η αποκοπή των δεδομένων σε μία χρονική στιγμή n, ώστε να χρησιμοποιηθούν ως πεπερασμένες σειρές:

$$u(n-p), u(n-p+1), \dots, u(n), d(n-p), d(n-p+1), \dots, d(n).$$

Εδώ, η υπολογιστική πολυπλοκότητα είναι $O(N^2)$, αφού το δ χρειάζεται να υπολογίζεται μόνο μία φορά κάθε χρονική στιγμή. Ένα πιθανό πρόβλημα της BPTT με περικοπή είναι ότι ενέργειες στη μνήμη που ξεπερνούν μία διάρκεια p δεν είναι αντιληπτές από το μοντέλο. Γενικά, η BPTT δυσκολεύεται να αντιληφθεί ενέργειες στη μνήμη με μακρά διάρκεια, καθώς οι πληροφορίες για την gradient του σφάλματος τείνουν να εξασθενίζουν σημαντικά με το χρόνο. Συνήθως, είναι δύσκολο να επιτευχθεί διάρκεια μνήμης που ξεπερνά τα 10 ή 20 χρονικά βήματα.

Η επαναλαμβανόμενη εκτέλεση των εποχών μεταβάλλει αργά ένα σύνθετο μη-γραμμικό δυναμικό σύστημα (το δίκτυο) σε παραμετρικό (βάρη) χώρο. Συνεπώς, όταν τα αρχικά βάρη προκαλούν μία ποιοτικά διαφορετική δυναμική συμπεριφορά από αυτή που απαιτείται, τότε εμφανίζονται διακλαδώσεις. Κοντά σε αυτές τις διακλαδώσεις, οι πληροφορίες της gradient δε χρησιμεύουν, καθώς επιβραδύνουν κατά πολύ τη σύγκλιση. Το σφάλμα μπορεί ακόμα και να αυξηθεί ξαφνικά στην περιοχή γύρω από αυτά τα κρίσιμα σημεία, λόγω της τομής με τα όρια της διακλάδωσης. Σε αντίθεση με την οπισθοδιάδοση στα δίκτυα πρόσθιας τροφοδότησης, τα οποία υλοποιούν συναρτήσεις και όχι δυναμικά συστήματα, η BPTT δεν εγγυάται τη σύγκλιση σε ένα τοπικό ελάχιστο σφάλματος. Επιπλέον, η BPTT κατά κανόνα χρησιμοποιείται σε δίκτυα μικρού μεγέθους της τάξης των 3 με 20 κόμβων, εξαιτίας του περιορισμένου υπολογιστικού χρόνου.

Συνολικά, χρειάζονται αρκετός υπολογιστικός χρόνος και δοκιμές της μεθόδου μέχρι την επίτευξη ενός καλού αποτελέσματος.

Real-Time Recurrent Learning: Μία περισσότερο υπολογιστικά ακριβή online παραλλαγή της BPTT είναι η RTRL. Η RTRL είναι μία μέθοδος κατάβασης κλίσης (gradient descent) η οποία υπολογίζει την ακριβή κλίση (gradient) του σφάλματος σε κάθε χρονικό βήμα και, άρα, είναι κατάλληλη για online εκπαιδευτικές διαδικασίες.

Η επίδραση της αλλαγής των βαρών στη δυναμική του δικτύου μπορεί να διαπιστωθεί παραγωγίζοντας τις εξισώσεις $(M1)$ και $(M2)$ ως προς τα βάρη τους. Χάριν ευκολίας, οι ενεργοποιήσεις όλων των κόμβων συμβολίζονται με v_i και τα βάρη με w_{kl} , με i=1, …, N για τους εσωτερικούς κόμβους, i=N+1, …, N+L για τους κόμβους εξόδου και i=N+L+1, …, N+L+K για τους κόμβους εισόδου. Η παράγωγος ενός εσωτερικού κόμβου ή ενός κόμβου εξόδου ως προς ένα βάρος w_{kl} είναι:

$$(R1) \quad \frac{\partial v_i(n+1)}{\partial w_{kl}} = f'(z_i(n)) \left[\left(\sum_{j=1}^{N+L} w_{ij} \frac{\partial v_j(n)}{\partial w_{kl}} \right) + \delta_{ik} v_l(n) \right], \text{με } i = 1, \dots, N+L,$$

όπου $k, l \leq N + L + K$, το $z_i(n)$ είναι πάλι η μέγιστη τιμή που μπορεί να πάρει ο αντίστοιχος κόμβος, αλλά το δ_{ik} εδώ συμβολίζει τη συνάρτηση δέλτα του Kronecker ($\delta_{ik} = 1$ αν $i = k$ και, διαφορετικά, $\delta_{ik} = 0$). Ο όρος $\delta_{ik} v_l(n)$ αναπαριστά την άμεση επίδραση του βάρους w_{kl} στον κόμβο k και το άθροισμα αναπαριστά την έμμεση επίδραση σε όλους τους κόμβους λόγω της δυναμικής του δικτύου. Η εξίσωση (R1) για κάθε εσωτερικό κόμβο ή κόμβο εξόδου συνιστά ένα γραμμικό δυναμικό σύστημα N+L διαστάσεων και διακριτού χρόνου με χρονικά μεταβαλλόμενους συντελεστές, όπου το

$$(R2) \quad \left(\frac{\partial v_1}{\partial w_{kl}}, \dots, \frac{\partial v_{N+L}}{\partial w_{kl}} \right)$$

θεωρείται ως μία δυναμική μεταβλητή. Δεδομένου ότι η αρχική κατάσταση του δικτύου είναι ανεξάρτητη των βαρών, αρχικοποιούμε την (R1) με:

$$\frac{\partial v_i(0)}{\partial w_{kl}} = 0 \, .$$

Με αυτό τον τρόπο, μπορούμε να υπολογίσουμε το (R2) προς τα εμπρός (forward) στο χρόνο επαναλαμβάνοντας την (R1) ταυτόχρονα με τις $(M1)$ και $(M2)$. Από τη λύση αυτή, μπορούμε να υπολογίσουμε την gradient του σφάλματος (για το σφάλμα της (B1)):

$$(R3)\quad \frac{\partial E}{\partial w_{kl}} = 2\sum_{n=1}^{T}\sum_{i=N}^{N+L}(v_i(n) - d_i(n))\frac{\partial v_i(n)}{\partial w_{kl}}.$$

Ένας κλασικός αλγόριθμος κατάβασης κλίσης που χρησιμοποιεί τη μέθοδο εκπαίδευσης με παρτίδες είναι η συγκέντρωση της κλίσης του σφάλματος με την εξίσωση (R3) και η ενημέρωση κάθε βάρους μετά την ολοκλήρωση μίας εποχής όπου παρουσιάζονται όλα τα δεδομένα εκπαίδευσης ως

$$new\ w_{kl} = w_{kl} - \gamma\frac{\partial E}{\partial w_{kl}},$$

όπου γ είναι ο ρυθμός εκμάθησης. Μία εναλλακτική ενημέρωση των βαρών αποτελεί η κατάβαση κλίσης του τρέχοντος σφάλματος εξόδου σε κάθε χρονικό βήμα:

$$w_{kl}(n+1) = w_{kl}(n) - \gamma\sum_{i=1}^{L}(v_i(n) - d_i(n))\frac{\partial v_i(n)}{\partial w_{kl}}.$$

Παραπάνω υποθέσαμε ότι το w_{kl} είναι σταθερή και όχι δυναμική μεταβλητή ώστε να εξάγουμε την (R1). Για αυτό το λόγο το γ πρέπει να διατηρηθεί σε χαμηλές κατάλληλες τιμές.

Η RTRL είναι σαφής από μαθηματική άποψη και θεωρητικά κατάλληλη για online εκπαίδευση. Ωστόσο, το υπολογιστικό κόστος είναι $O((N+L)^4)$ για κάθε βήμα ενημέρωσης, αφού πρέπει να λύνουμε το σύστημα (R1) το οποίο είναι N+L διαστάσεων για κάθε βάρος. Έτσι, η RTRL είναι χρήσιμη για online προσαρμογή μόνο στην περίπτωση που πολύ μικρά δίκτυα αρκούν.

Extended Kalman Filter: Το φίλτρο Kalman είναι ένας αλγόριθμος ο οποίος χρησιμοποιείται σε γραμμικά μοντέλα δυναμικών συστημάτων που δέχονται εξωτερικές φυσικές διαταραχές/θόρυβο. Στόχος του είναι η εξαγωγή εκτιμήσεων της κατάστασης του συστήματος χωρίς διαταραχές. Εντούτοις, αν το μοντέλο είναι μη γραμμικό, τότε ο αλγόριθμος μπορεί να επεκταθεί σε μία διαδικασία γραμμικοποίησης και το φίλτρο που προκύπτει ονομάζεται extended Kalman filter. Ας θεωρήσουμε μία εύκολη ειδική περίπτωση, ένα σύστημα διακριτού χρόνου με πρόσθετη είσοδο και καθόλου θόρυβο παρατήρησης:

$$(K1)\quad x(n+1) = f\big(x(n)\big) + q(n),$$

$$(K2)\quad d(n) = h_n(x(n)),$$

όπου $x(n)$ είναι το διάνυσμα της εσωτερικής κατάστασης του συστήματος, f είναι η συνάρτηση ενημέρωσης της κατάστασης του συστήματος (η οποία είναι γραμμική στο απλό φίλτρο Kalman), $q(n)$ είναι εξωτερική είσοδος στο σύστημα (μία διαδικασία ασυσχέτιστου Γκαουσιανού λευκού θορύβου[17] μπορεί, επίσης, να θεωρηθεί θόρυβος), $d(n)$ είναι η έξοδος του συστήματος και h_n είναι μία χρονικά εξαρτώμενη συνάρτηση παρατήρησης (η οποία είναι γραμμική στο απλό φίλτρο Kalman). Τη χρονική στιγμή $n = 0$, μία πολυδιάστατη κανονική κατανομή με μέση τιμή $\hat{x}(0)$ και πίνακα συνδιακύμανσης[18] $P(0)$ «μαντεύει» την κατάσταση του συστήματος $x(0)$. Μέχρι τη χρονική στιγμή n το σύστημα παρατηρείται μέσω των $d(0), \dots, d(n)$. Αυτό που κάνει το EKF είναι να δώσει μία εκτίμηση $\hat{x}(n+1)$ της πραγματικής κατάστασης $x(n+1)$, δοθέντων της «μαντεψιάς» της αρχικής κατάστασης και όλων των

[17] Στο λευκό Γκαουσιανό θόρυβο, οι τιμές είναι ταυτοτικά κατανεμημένες και στατιστικά ανεξάρτητες (και άρα ασυσχέτιστες).

[18] Πίνακας συνδιακύμανσης είναι ένας τετραγωνικός πίνακας ο οποίος μας δίνει τη συνδιακύμανση κάθε ζεύγους στοιχείων ενός δοθέντος τυχαίου διανύσματος.

προηγούμενων παρατηρήσεων εξόδου. Για να επιτευχθεί αυτό γίνονται οι παρακάτω υπολογισμοί:

Πρόβλεψη $(K3)$

$$\hat{x}^*(n) = F(\hat{x}(n))$$

$$P^*(n) = F(n)P(n-1)F(n)^t + Q(n)$$

Ενημέρωση $(K4)$

$$K(n) = P^*(n)H(n)[H(n)^t P^*(n)H(n)]^{-1}$$

$$\hat{x}(n+1) = \hat{x}^*(n) + K(n)\xi(n)$$

$$P(n+1) = P^*(n) - K(n)H(n)^t P^*(n)$$

όπου τα $F(n)$ και $H(n)$ είναι οι ιακωβιανοί πίνακες των συναρτήσεων f και h_n ως προς τις μεταβλητές κατάστασης, που έχουν υπολογιστεί στην εκτίμηση της προηγούμενης κατάστασης:

$$F(n) = \frac{\partial f(x)}{\partial x}\Big|_{x=\hat{x}(n)} \quad , \quad H(n) = \frac{\partial h_n(x)}{\partial x}\Big|_{x=\hat{x}(n)}$$

Επίσης, όπου $\xi(n) = d(n) - h_n(\hat{x}(n))$ είναι το σφάλμα (δηλαδή η διαφορά μεταξύ της παρατηρούμενης εξόδου και της εξόδου που υπολογίζεται από την εκτίμηση της κατάστασης $\hat{x}(n)$), $P(n)$ είναι μία εκτίμηση του πίνακα συνδιακύμανσης του δεσμευμένου σφάλματος $E[\xi\xi|d(0), \dots, d(n)]$, $Q(n)$ είναι ο διαγώνιος πίνακας συνδιακύμανσης του θορύβου. Τέλος, τα $\hat{x}^*(n)$ (εκτίμηση κατάστασης του συστήματος) και $P^*(n)$ (εκτίμηση συνδιακύμανσης του σφάλματος της κατάστασης) προκύπτουν από την προσθήκη των δυναμικών f στις προηγούμενες εκτιμήσεις.

Η βασική ιδέα του φίλτρου Kalman είναι να γίνει ενημέρωση των $\hat{x}(n)$, $P(n)$ με κάποιες αρχικές προβλέψεις $\hat{x}^*(n)$, $P^*(n)$ οι οποίες συμπεραίνονται από τις προηγούμενές τους τιμές, εφαρμόζοντας τα γνωστά δυναμικά στους υπολογισμούς (K3) και, έπειτα, προσαρμόζοντας αυτές τις αρχικές προβλέψεις με την ενσωμάτωση των πληροφοριών που περιέχονται στο $d(n)$. Έτσι, στην (K4) σχηματίζεται το κέρδος Kalman (Kalman gain) $K(n)$. Στο απλό, γραμμικό, φίλτρο Kalman, τα $F(n)$ και $H(n)$ είναι σταθερά και οι εκτιμήσεις της κατάστασης συγκλίνουν στην πραγματική δεσμευμένη μέση τιμή [19] των καταστάσεων $E[x(n)|d(0), \dots, d(n)]$. Για μη γραμμικές f και h_n, δεν ισχύει γενικά αυτό και η χρήση των EKFs έχει ως αποτέλεσμα μόνο τοπικά βέλτιστες εκτιμήσεις κατάστασης.

Όσον αφορά στην εφαρμογή των EKF στην εκτίμηση βαρών σε επαναλαμβανόμενα νευρωνικά δίκτυα, ας θεωρήσουμε ότι υπάρχει ένα επαναλαμβανόμενο νευρωνικό δίκτυο το οποίο αναπαράγει τέλεια τις χρονικές σειρές εισόδου/εξόδου των δεδομένων εκπαίδευσης:

$$u(n) = \left(u_1(n), \dots, u_K(n)\right)^t, d(n) = \left(d_1(n), \dots, d_L(n)\right)^t, n = 1, \dots, T,$$

όπου τα βάρη των συνδέσεων εισόδου/εσωτερικών/εξόδου/εξόδου πίσω με τις εσωτερικές συνδέσεις συγκεντρώνονται στους αντίστοιχους πίνακες βαρών $N \times K/\ N \times N/\ L \times (K + N + L)/\ N \times L$ διαστάσεων:

[19] Δεσμευμένη μέση τιμή μίας τυχαίας μεταβλητής X είναι η αναμενόμενη τιμή της X, δηλαδή η τιμή που θα έπαιρνε κατά μέσο όρο μετά από έναν αυθαίρετα μεγάλο αριθμό συμβάντων, δεδομένου ότι είναι γνωστό πως κάποιες συγκεκριμένες «δεσμεύσεις» Y θα εμφανιστούν. Συμβολίζεται με $E[X|Y]$.

$W^{in} = (w_{ij}^{in}), W = (w_{ij}), W^{out} = (w_{ij}^{out}), W^{back} = (w_{ij}^{back})$.

Οι διαφορετικοί αυτοί τύποι βαρών δε θα διαχωρίζονται παρακάτω, αλλά θα θεωρούνται όλοι ως ένα διάνυσμα βάρους w.

Για να εφαρμόσουμε, λοιπόν, τα EKF στη διαδικασία της εκτίμησης βέλτιστων βαρών ενός επαναλαμβανόμενου νευρωνικού δικτύου, εκλαμβάνουμε τα βάρη w του τέλειου επαναλαμβανόμενου νευρωνικού δικτύου ως την κατάσταση ενός δυναμικού συστήματος. Η έξοδος $d(n)$ του επαναλαμβανόμενου νευρωνικού δικτύου είναι μία συνάρτηση h των βαρών και της εισόδου έως τη χρονική στιγμή n, όπου θεωρούμε ότι οι παροδικές επιδράσεις της αρχικής κατάστασης του δικτύου έχουν χαθεί:

$d(n) = h(w, u(0), \dots, u(n))$.

Οι είσοδοι μπορούν να ενσωματωθούν στη συνάρτηση εξόδου h, κάνοντάς τη μία χρονικά εξαρτώμενη συνάρτηση h_n. Επιπλέον, θεωρούμε ότι η ενημέρωση του δικτύου περιέχει κάποιο θόρυβο, τον οποίο προσθέτουμε στα βάρη με τη μορφή ενός Γκαουσιανού ασυσχέτιστου θορύβου $q(n)$. Με την προσθήκη αυτή λέγεται ότι βελτιώνεται η αριθμητική ευστάθεια του αλγορίθμου. Έτσι, προκύπτει η παρακάτω εκδοχή των $(K1)$ και $(K2)$ για τη δυναμική του τέλειου επαναλαμβανόμενου νευρωνικού δικτύου:

$(K5)$ $w(n+1) = w(n) + q(n)$,

$(K6)$ $d(n) = h_n(w(n))$.

Εκτός από τις μετατοπίσεις που προκλήθηκαν από το θόρυβο, η δυναμική της κατάστασης του συστήματος είναι στατική και η είσοδος $u(n)$ στο δίκτυο δεν εισέρχεται στην εξίσωση ενημέρωσης της κατάστασης, αλλά είναι κρυμμένη στη χρονική εξάρτηση της συνάρτησης παρατήρησης.

Τώρα, η εκπαιδευτική διαδικασία του δικτύου παίρνει τη μορφή της εκτίμησης της στατικής κατάστασης $w(n)$ με βάση μία αρχική «μαντεψιά» $\widehat{w}(0)$ και την ακολουθία των εξόδων $d(0), \dots, d(n)$. Ο πίνακας συνδιακύμανσης του σφάλματος $P(0)$ αρχικοποιείται ως ένας διαγώνιος πίνακας με μεγάλες τιμές στα στοιχεία της κύριας διαγωνίου, για παράδειγμα 100. Η απλούστερη μορφή των $(K5)$ και $(K6)$ από τις $(K1)$ και $(K2)$, αντίστοιχα, οδηγεί σε μερικές απλοποιήσεις των EKF υπολογισμών $(K3)$ και $(K4)$. Οι υπολογισμοί της πρόβλεψης δεν είναι πλέον απαραίτητοι, αφού η δυναμική της κατάστασης του συστήματος, δηλαδή το βάρος, είναι τετριμμένη. Οι υπολογισμοί της ενημέρωσης γίνονται:

$$K(n) = P(n)H(n)[H(n)^t P(n)H(n)]^{-1}$$

$(K7)$ $$\widehat{w}(n+1) = \widehat{w}(n) + K(n)\xi(n)$$

$$P(n+1) = P(n) - K(n)H(n)^t P(n) + Q(n)$$

Ένας ρυθμός εκμάθησης η μπορεί να εισαχθεί στην εξίσωση του κέρδους του Kalman, ο οποίος στην αρχή της εκπαίδευσης θα είναι μικρός για να αντισταθμίσει τις αρχικές λανθασμένες εκτιμήσεις του $P(n)$:

$K(n) = P(n)H(n)[(1/\eta)I + H(n)^t P(n)H(n)]^{-1}$.

Το EKF είναι ένας αλγόριθμος κατάβασης κλίσης δεύτερου βαθμού που αξιοποιεί τις πληροφορίες για την καμπυλότητα της επιφάνειας του τετραγωνικού σφάλματος. Απόρροια αυτού είναι η ικανότητα του φίλτρου Kalman να συγκλίνει με ένα και μόνο βήμα στα γραμμικά χωρίς θόρυβο συστήματα. Ας το παρουσιάσουμε, λοιπόν, με ένα απλό παράδειγμα δικτύου πρόσθιας τροφοδότησης. Θεωρούμε το εξής δίκτυο που αποτελείται από μία είσοδο

και μία έξοδο και το οποίο συνδέει τους κόμβους εισόδου με τους κόμβους εξόδου με μία σύνδεση με βάρος w, χωρίς εσωτερικούς κόμβους:

$$w(n+1) = w(n),$$

$$d(n) = w\,u(n).$$

Εξετάζουμε το εκτελούμενο EKF (στην εκδοχή των (K7)) για κάποια χρονική στιγμή n, όπου έχει φτάσει μία εκτίμηση $\hat{w}(n)$. Παρατηρώντας ότι ο ιακωβιανός $H(n)$ είναι $dwu(n)/dw = u(n)$, η επόμενη εκτιμώμενη κατάσταση είναι:

$$\hat{w}(n+1) = \hat{w}(n) + K(n)\xi(n) = \hat{w}(n) + \frac{1}{u(n)}\big(wu(n) - \hat{w}u(n)\big) = w.$$

Από το παράδειγμα αυτό γίνεται αντιληπτό ότι το EKF εμφανίζει γρήγορη σύγκλιση, τουλάχιστον για περιπτώσεις στις οποίες η τρέχουσα εκτίμηση $\hat{w}(n)$ είναι ήδη κοντά στη σωστή τιμή, έτσι ώστε η γραμμικοποίηση να προσφέρει μία καλή προσέγγιση στο πραγματικό σύστημα.

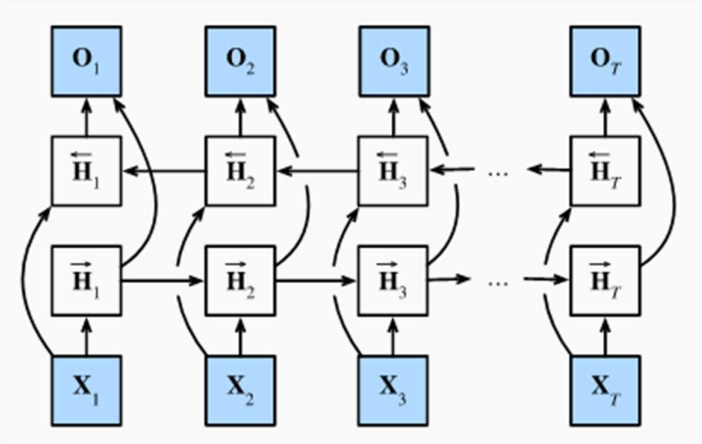
Το EKF απαιτεί τις παραγώγους $H(n)$ των εξόδων του δικτύου ως προς τα βάρη που έχουν υπολογιστεί στην τρέχουσα εκτίμηση βαρών. Οι παράγωγοι αυτές μπορούν να υπολογιστούν ακριβώς όπως και στον αλγόριθμο της RTRL, με υπολογιστικό κόστος $O(N^4)$. Συνεπώς, είναι πολύ υπολογιστικά ακριβό, εκτός και αν το δίκτυο στο οποίο πρέπει να εφαρμοστεί είναι μικρό. Εναλλακτικά, μία διέξοδος είναι η BPTT με αποκοπή, με χρήση μίας εκδοχής της $(K6)$ με πεπερασμένη ακολουθία εξόδων αντί μίας εξόδου και λαμβάνοντας προσεγγίσεις του $H(n)$ με μία διαδικασία ανάλογη των $(B2)$ έως και $(B5)$. Εδώ, το υπολογιστικό κόστος είναι $O(pN^2)$, όπου p είναι το βάθος της αποκοπής. Εκτός από τον υπολογισμό του $H(n)$, η πιο ακριβή διαδικασία στο EKF είναι η ενημέρωση του $P(n)$, με κόστος $O(LN^2)$.

Εν κατακλείδι, τα καλύτερα αποτελέσματα όσον αφορά στην εκπαίδευση των επαναλαμβανόμενων νευρωνικών δικτύων φαίνεται πως επιτυγχάνονται με εφαρμογή του EKF, χρησιμοποιώντας BPTT με αποκοπή για την εκτίμηση του $H(n)$. Η συνολική επιτυχία και ποιότητα της εκπαίδευσης με χρήση του EKF βασίζεται στην εμπειρία, η οποία οδηγεί στην κατάλληλη επιλογή της αρχιτεκτονικής του δικτύου, των ρυθμών εκμάθησης, των λεπτομερειών στους υπολογισμούς gradient κτλ.

3.3 Επεκτάσεις των ΕΝΔ

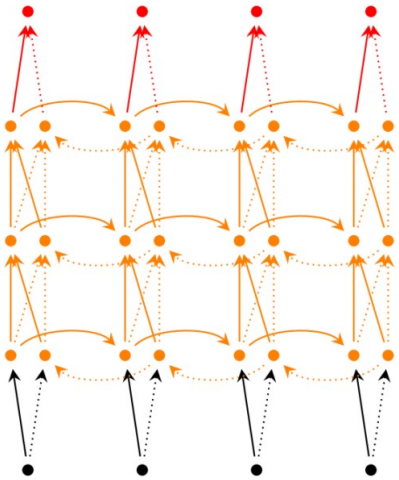
Κάποιοι περιορισμοί των αρχικών επαναλαμβανόμενων νευρωνικών δικτύων οδήγησαν στην εύρεση πιο αναπτυγμένων μορφών, μερικοί εκ των οποίων είναι οι ακόλουθοι:

Αμφίδρομα (Bidirectional) επαναλαμβανόμενα νευρωνικά δίκτυα: Συνδυάζουν ένα επαναλαμβανόμενο νευρωνικό δίκτυο που κινείται προς τα εμπρός στο χρόνο, το οποίο ξεκινά από την αρχή της ακολουθίας, με ένα άλλο επαναλαμβανόμενο νευρωνικό δίκτυο που κινείται προς τα πίσω στο χρόνο, το οποίο ξεκινά από το τέλος της ακολουθίας. Αυτό επιτρέπει στις μονάδες εξόδου να υπολογίσουν μία αναπαράσταση που βασίζεται και στο παρελθόν και στο μέλλον, αλλά είναι πιο ευαίσθητη στις τιμές εισόδου την παρούσα χρονική στιγμή t. Γενικά, έχει παρατηρηθεί ότι δίνουν καλύτερα αποτελέσματα σε εργασίες ταξινόμησης (classification), όπως το πρόγραμμα της παρούσας διπλωματικής εργασίας, και παλινδρόμησης (regression).



Εικόνα 16: Η δομή ενός αμφίδρομου RNN.[20]

Βαθιά Αμφίδρομα (Deep Bidirectional) επαναλαμβανόμενα νευρωνικά δίκτυα: Όμοια με τα αμφίδρομα επαναλαμβανόμενα νευρωνικά δίκτυα, με τη διαφορά ότι τώρα έχουμε πολλαπλά επίπεδα σε κάθε χρονικό βήμα. Αυτό μας δίνει υψηλότερη μαθησιακή ικανότητα, αλλά χρειαζόμαστε περισσότερα δεδομένα εκπαίδευσης.



Εικόνα 17: Η δομή ενός βαθιά αμφίδρομου RNN.[21]

Δίκτυα μακράς βραχύχρονης μνήμης (Long Short-Term Memory): Χρησιμοποιούν μια διαφορετική συνάρτηση για να υπολογίσουν την κρυφή κατάσταση. Τα δίκτυα μακράς βραχύχρονης μνήμης είναι πολύ αποτελεσματικά όσον αφορά στις μακροπρόθεσμες εξαρτήσεις. Παρακάτω θα αναλύσουμε επαρκώς τον τρόπο λειτουργίας τους.

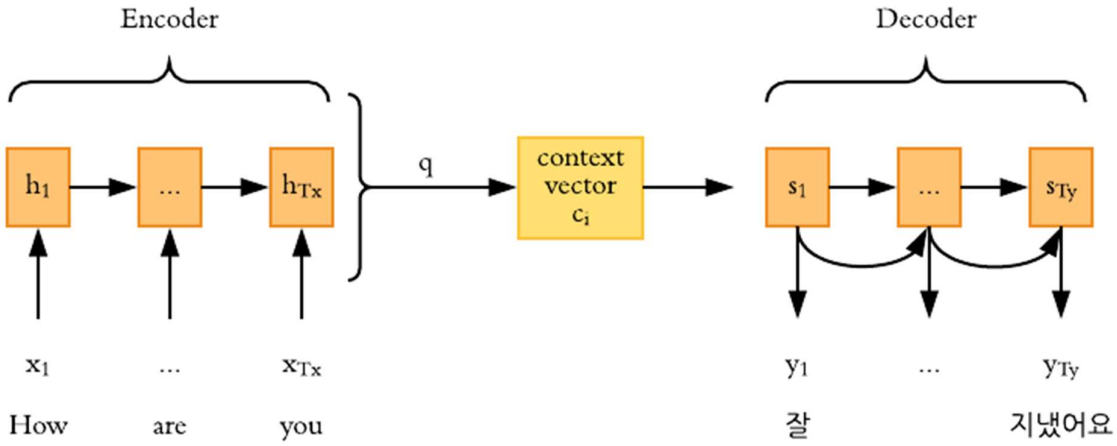
3.4 Εφαρμογές των ΕΝΔ

Τα επαναλαμβανόμενα νευρωνικά δίκτυα χρησιμοποιούνται ευρέως σε διάφορους τομείς. Ένας στόχος της βαθιάς μάθησης (deep learning) είναι η ανάπτυξη αλγορίθμων ικανών να επιλύουν μία μεγάλη γκάμα εργασιών. Ωστόσο, μέχρι στιγμής χρειάζεται κάποια εξειδίκευση του αλγορίθμου ανάλογα τον τομέα. Για παράδειγμα, εργασίες σχετικές με τη μηχανική όραση

[20] Πηγή: https://d2l.ai/chapter_recurrent-modern/bi-rnn.html .
[21] Πηγή: https://ainewgeneration.com/recurrent-neural-network/ .

απαιτούν την επεξεργασία μεγάλου αριθμού χαρακτηριστικών εισόδου (pixels) για κάθε παράδειγμα. Ακόμη, εργασίες σχετικές με τη γλώσσα απαιτούν τη μοντελοποίηση μεγάλου αριθμού πιθανών τιμών (λέξεις στο λεξιλόγιο) για κάθε χαρακτηριστικό εισόδου. Κάποιες από τις εφαρμογές των επαναλαμβανόμενων νευρωνικών δικτύων είναι οι ακόλουθες:

- Μοντελοποίηση γλώσσας και παραγωγή κειμένου: Λαμβάνοντας ως είσοδο μία ακολουθία λέξεων, προσπαθούμε να προβλέψουμε την πιθανότητα της επόμενης λέξης. Κατά την εκπαίδευση, θεωρούμε $x_{t+1} = o_t$, δηλαδή η έξοδος της τρέχουσας χρονικής στιγμής θα είναι η είσοδος της επόμενης χρονικής στιγμής. Η μέθοδος αυτή μπορεί να θεωρηθεί ως μία από τις πιο χρήσιμες προσεγγίσεις για μετάφραση, καθώς η πιο πιθανή πρόταση θα είναι και η σωστή.
- Μηχανική μετάφραση: Στη μηχανική μάθηση, η είσοδος είναι μία ακολουθία λέξεων σε κάποια γλώσσα, π.χ. στην Κινεζική, και η έξοδος είναι αυτή η ακολουθία λέξεων μεταφρασμένη στην επιθυμητή γλώσσα, π.χ. στην Ελληνική. Η κύρια διαφορά της μηχανικής μάθησης με τη μοντελοποίηση γλώσσας είναι ότι το δίκτυο εξάγει το αποτέλεσμα, δηλαδή το μεταφρασμένο κείμενο, αφού λάβει ολόκληρη την είσοδο, δηλαδή ολόκληρο το κείμενο που πρέπει να μεταφραστεί. Σχεδόν όλα τα συστήματα μετάφρασης που χρησιμοποιούνται σήμερα χρησιμοποιούν επαναλαμβανόμενα νευρωνικά δίκτυα.



Εικόνα 18: Μηχανική μετάφραση με είσοδο μία ακολουθία αγγλικών λέξεων και έξοδο τη μετάφραση της πρότασης αυτής στην Κορεατική γλώσσα. Το q είναι μία συνάρτηση η οποία παράγει το διάνυσμα συγκείμενου των κρυφών καταστάσεων κατά την κωδικοποίηση (encoding).[22]

- Αναγνώριση ομιλίας: Το σύνολο των δεδομένων εισόδου αποτελείται από φωνήματα[23] ή ακουστικά σήματα ενός ήχου. Τα δεδομένα αυτά επεξεργάζονται με κατάλληλο τρόπο ώστε να αποτελούν μία συλλογή ηχητικών κυμάτων. Τα ηχητικά κύματα ταξινομούνται σε πολύ μικρά φωνητικά τμήματα (segments) και ενώνονται σε συνεκτικές λέξεις με εφαρμογή ενός επαναλαμβανόμενου νευρωνικού δικτύου. Η έξοδος αποτελείται από ένα μοτίβο φωνητικών τμημάτων ενωμένων σε ένα σύνολο με λογικό τρόπο μαζί με τις πιθανότητές τους.
- Δημιουργία περιγραφών εικόνων: Ένας συνδυασμός συνελικτικών νευρωνικών δικτύων [24] (convolutional neural networks ή CNNs) και επαναλαμβανόμενων

[22] Πηγή: https://laptrinhx.com/attention-in-nlp-314116443/ .
[23] Τα φωνήματα αποτελούν τα ελάχιστα στοιχεία μίας γλώσσας, που παρέχουν διαφοροποιητική λειτουργία στο φωνητικό επίπεδο, για το νόημα του γλωσσικού ήχου.
[24] Συνελικτικά νευρωνικά δίκτυα είναι νευρωνικά δίκτυα που εφαρμόζονται σε δισδιάστατους πίνακες, κυρίως εικόνες. Εντοπίζουν χαρακτηριστικά εικόνων βάσει των οποίων, έπειτα, εντοπίζουν πιο αφηρημένα χαρακτηριστικά.

νευρωνικών δικτύων χρησιμοποιούνται για τη δημιουργία μίας περιγραφής του τι υπάρχει σε μία εικόνα. Το συνελικτικό νευρωνικό δίκτυο κάνει το διαχωρισμό σε μικρά τμήματα και το επαναλαμβανόμενο νευρωνικό δίκτυο χρησιμοποιεί αυτά τα τμηματοποιημένα δεδομένα για να σχηματίσει την περιγραφή.

Άλλες εφαρμογές των επαναλαμβανόμενων νευρωνικών δικτύων περιλαμβάνουν τη σύνθεση μουσικής, την αναγνώριση χειρόγραφου κειμένου, την εκμάθηση γραμματικών κανόνων, την περίληψη κειμένων κτλ. Αποδεικνύεται έτσι η τεράστια σημασία τους στις μέρες μας.

3.5 Το Πρόβλημα των Μακροχρόνιων Εξαρτήσεων

Ένα από τα πολύ χρήσιμα χαρακτηριστικά των επαναλαμβανόμενων νευρωνικών δικτύων είναι ότι μπορούν να συνδέσουν προηγούμενες πληροφορίες με την τωρινή εκπαίδευση. Ωστόσο, αυτό δεν είναι πάντα δυνατό. Μερικές φορές, χρειαζόμαστε μόνο τις πρόσφατες πληροφορίες για να εκτελέσουμε την τρέχουσα εκπαίδευση. Για παράδειγμα, ας θεωρήσουμε ένα γλωσσικό μοντέλο που προσπαθεί να προβλέψει την επόμενη λέξη βάσει των προηγούμενων λέξεων. Αν θέλουμε να προβλέψουμε την τελευταία λέξη στην πρόταση «η Γη είναι πλανήτης», όπου δηλαδή το κενό μεταξύ της πληροφορίας που μας δίνεται και αυτού που ψάχνουμε είναι μικρό, το επαναλαμβανόμενο νευρωνικό δίκτυο μπορεί να μάθει να χρησιμοποιεί την προηγούμενη πληροφορία και να βγάλει το σωστό αποτέλεσμα. Από την άλλη, αν θέλουμε να προβλέψουμε την τελευταία λέξη στο «Μεγάλωσα στην Αγγλία... Μιλάω άπταιστα αγγλικά», οι πρόσφατες πληροφορίες που δίνονται δείχνουν ότι η επόμενη λέξη είναι μάλλον το όνομα μίας γλώσσας. Όμως, αν θέλουμε να περιορίσουμε τις γλώσσες, στη μία από τις οποίες αναφέρεται το κείμενο, χρειαζόμαστε την πληροφορία «Αγγλία» από πιο πίσω. Είναι εξαιρετικά πιθανό το κενό μεταξύ της πληροφορίας που δίνεται και του σημείου στο οποίο θα χρειάζεται να γίνει πολύ μεγάλο. Δυστυχώς, όσο το κενό μεγαλώνει, τα επαναλαμβανόμενα νευρωνικά δίκτυα δεν μπορούν να μάθουν να συνδέουν τις πληροφορίες μεταξύ τους.

Τα επαναλαμβανόμενα νευρωνικά δίκτυα κατασκευάζουν πολύ βαθιά υπολογιστικά γραφήματα (computational graphs), εφαρμόζοντας επανειλημμένως την ίδια διαδικασία σε κάθε χρονικό βήμα μιας μεγάλης χρονικής ακολουθίας. Επαναλαμβανόμενη εφαρμογή των ίδιων παραμέτρων γεννά ιδιαίτερα έντονες δυσκολίες.

Το βασικό πρόβλημα είναι ότι κλίσεις (gradients) που διαδίδονται σε πολλά επίπεδα τείνουν, τις περισσότερες φορές, να εξαφανίζονται (vanishing) ή, σπάνια, να εκρήγνυνται (explosion).

- Πρόβλημα εξαφάνισης των κλίσεων (gradients): Οι κλίσεις οι οποίες χρησιμοποιούνται για την ενημέρωση των βαρών μικραίνουν εκθετικά γρήγορα και τελικά εξαφανίζονται έπειτα από μερικά βήματα. Το αποτέλεσμα είναι τα βάρη να μην ανανεώνονται πια και να εμποδίζεται η εκπαίδευση του δικτύου, όπως περιγράψαμε στην ενότητα των συναρτήσεων ενεργοποίησης παραπάνω.
- Πρόβλημα έκρηξης των κλίσεων (gradients): Οι κλίσεις οι οποίες χρησιμοποιούνται για την ενημέρωση των βαρών μεγαλώνουν εκθετικά γρήγορα. Αυτό εμποδίζει τον αλγόριθμο οπισθοδιάδοσης να ενημερώσει με λογικό τρόπο τα βάρη και η εκπαίδευση γίνεται ασταθής.

Το πρόβλημα της εξαφάνισης των κλίσεων (gradients) έχει απασχολήσει περισσότερο τους ερευνητές από το πρόβλημα της έκρηξης, διότι δεν είναι φανερό πότε συμβαίνει και πώς επιλύεται. Αντίθετα, κατά την έκρηξη, οι κλίσεις παίρνουν την τιμή NaN (not a number) και το πρόγραμμα σταματά. Επιπλέον, υπάρχει η απλή και αποτελεσματική λύση του περιορισμού των κλίσεων από ένα προκαθορισμένο κατώφλι.

Ακόμα και αν υποθέσουμε ότι οι παράμετροι είναι τέτοιες ώστε το επαναλαμβανόμενο νευρωνικό δίκτυο να είναι σταθερό (δηλαδή να μπορεί να αποθηκεύει πληροφορίες στη μνήμη χωρίς να εκρήγνυνται οι κλίσεις), η δυσκολία με τις μακροπρόθεσμες εξαρτήσεις προκύπτει

από τα εκθετικά μικρότερα βάρη (weights) που δίνονται στις μακροπρόθεσμες αλληλεπιδράσεις συγκριτικά με τις βραχυπρόθεσμες. Στην πράξη, τα πειράματα του Yoshua Bengio και άλλων ερευνητών περί το 1994, δείχνουν ότι όσο αυξάνεται το διάστημα των εξαρτήσεων που χρειάζονται να ληφθούν, τόσο όλο και περισσότερο δύσκολη γίνεται η βελτιστοποίηση (optimization) που βασίζεται στην κλίση, με την πιθανότητα επιτυχούς εκπαίδευσης (training) ενός παραδοσιακού επαναλαμβανόμενου νευρωνικού δικτύου μέσω στοχαστικής κλίσης κατάβασης (stochastic gradient descent) να φτάνει γρήγορα στο μηδέν για ακολουθίες μήκους μόνο 10 ή 20.

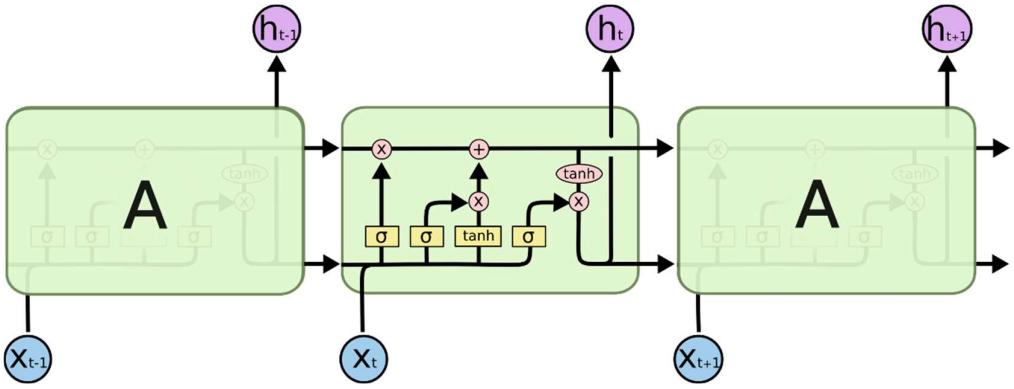
Ευτυχώς, υπάρχουν κάποιοι τρόποι αντιμετώπισης του προβλήματος της εξαφάνισης των κλίσεων (gradients). Καταρχάς, με κατάλληλη αρχικοποίηση του πίνακα των βαρών είναι δυνατό να μειωθεί η επίδραση της εξαφάνισης των κλίσεων. Τη δυνατότητα αυτή έχει και η εξομάλυνση. Ακόμη, μία προτιμότερη λύση είναι η χρήση της συνάρτησης ενεργοποίησης ReLU, όπως αναφέρθηκε σε παραπάνω ενότητα. Τέλος, εξαιρετικά δημοφιλής είναι η χρήση αρχιτεκτονικών δικτύων μακράς βραχύχρονης μνήμης (Long Short-Term Memory) ή gated recurrent unit (GRU). Και οι δύο αυτές αρχιτεκτονικές σχεδιάστηκαν για την επίλυση αυτού του προβλήματος και την εκμάθηση μακροχρόνιων εξαρτήσεων.

ΚΕΦΑΛΑΙΟ 4

ΔΙΚΤΥΑ ΜΑΚΡΑΣ ΒΡΑΧΥΧΡΟΝΗΣ ΜΝΗΜΗΣ (ΜΒΜ)

Στα επαναλαμβανόμενα νευρωνικά δίκτυα (Recurrent Neural Networks) είναι εφικτό να ενσωματώσουμε μία αρχιτεκτονική γνωστή ως δίκτυα μακράς βραχύχρονης μνήμης (Long Short-Term Memory ή LSTM). Τα δίκτυα μακράς βραχύχρονης μνήμης είναι ένα ιδιαίτερο είδος επαναλαμβανόμενου νευρωνικού δικτύου ικανό να μαθαίνει μακροχρόνιες εξαρτήσεις, να θυμάται δηλαδή πληροφορίες για μεγάλες περιόδους του χρόνου. Εισήχθησαν από τους Hochreiter και Schmidhuber το 1997 με ξεκάθαρο σκοπό τη βοήθεια στην αντιμετώπιση του προβλήματος της ασταθούς κλίσης (gradient). Κατόπιν, βελτιώθηκαν και διαδόθηκαν από πολλούς μεταγενέστερους ερευνητές. Δουλεύουν εξαιρετικά καλά σε μεγάλη ποικιλία προβλημάτων και σήμερα χρησιμοποιούνται ευρέως.

Τα δίκτυα μακράς βραχύχρονης μνήμης έχουν αλυσιδωτή δομή όπως τα επαναλαμβανόμενα νευρωνικά δίκτυα, αλλά η επαναλαμβανόμενη μονάδα (module) έχει διαφορετική δομή. Αντί να έχει ένα μοναδικό επίπεδο νευρωνικού δικτύου, έχει τέσσερα τα οποία αλληλεπιδρούν μεταξύ τους με έναν ιδιαίτερο τρόπο.



Εικόνα 19: Η δομή ενός LSTM.[25]

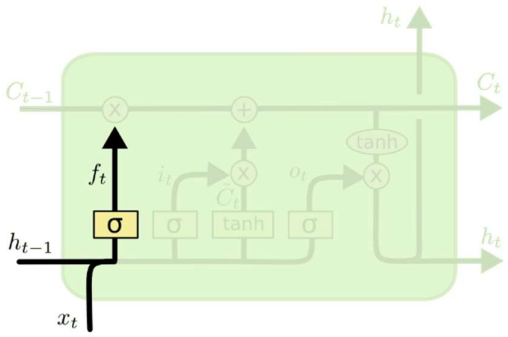
Στην παραπάνω εικόνα, κάθε γραμμή μεταφέρει ένα διάνυσμα από την έξοδο του ενός κόμβου στις εισόδους των άλλων κόμβων. Οι ροζ κύκλοι αναπαριστούν πράξεις κατά σημείο (pointwise), όπως η πρόσθεση διανυσμάτων. Τα κίτρινα πλαίσια είναι εκπαιδευμένα επίπεδα νευρωνικού δικτύου. Οι γραμμές που ενώνονται δηλώνουν σύνδεση, ενώ μία γραμμή που διακλαδώνεται δηλώνει ότι το περιεχόμενό της έχει αντιγραφεί και τα αντίγραφα πηγαίνουν σε διαφορετικές τοποθεσίες.

Το «κλειδί» στα δίκτυα μακράς βραχύχρονης μνήμης είναι το κελί (cell), δηλαδή η οριζόντια γραμμή στο πάνω μέρος του διαγράμματος. Το κελί είναι κάτι σαν ζώνη μεταφοράς. Διατρέχει ολόκληρη την αλυσίδα με μικρής σημασίας μόνο γραμμικές αλληλεπιδράσεις με τα υπόλοιπα στοιχεία. Είναι πολύ εύκολο για τις πληροφορίες να κυλούν κατά μήκος του χωρίς να αλλάζουν. Τα δίκτυα μακράς βραχύχρονης μνήμης μπορούν να αφαιρούν ή να προσθέτουν πληροφορίες στο κελί, οι οποίες έχουν επιλεγεί από δομές που ονομάζονται πύλες (gates). Οι πύλες αποτελούνται από ένα επίπεδο νευρωνικού δικτύου σιγμοειδούς συνάρτησης και μία πράξη πολλαπλασιασμού κατά σημείο. Το επίπεδο της σιγμοειδούς συνάρτησης βγάζει ως έξοδο αριθμούς μεταξύ του 0 και του 1, οι οποίοι περιγράφουν πόση πληροφορία από κάθε στοιχείο θα περάσει μέσα. Ο αριθμός 0 δηλώνει ότι δεν θα περάσει τίποτα, ενώ ο αριθμός 1

[25] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .

δηλώνει ότι θα περάσουν όλα. Ένα δίκτυο μακράς βραχύχρονης μνήμης έχει τρεις από αυτές τις πύλες για να προστατεύει και να ελέγχει το κελί.

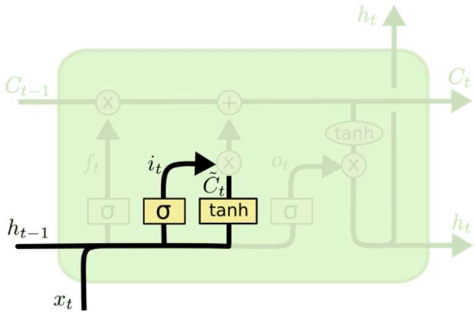
Το πρώτο βήμα στο δίκτυο μακράς βραχύχρονης μνήμης είναι να αποφασίσει ποιες πληροφορίες δε θα περάσουν στο κελί. Αυτή η απόφαση λαμβάνεται από ένα σιγμοειδές επίπεδο που ονομάζεται επίπεδο της πύλης της λήθης (forget gate layer). Εξετάζει τα h_{t-1} και x_t και βγάζει ως έξοδο έναν αριθμό μεταξύ του 0 και 1 για κάθε αριθμό στο κελί C_{t-1}. Ας επιστρέψουμε στο παράδειγμα που δώσαμε παραπάνω, για το γλωσσικό μοντέλο που προσπαθεί να προβλέψει την επόμενη λέξη βάσει των προηγούμενων λέξεων. Σε ένα τέτοιο πρόβλημα, το κελί μπορεί να περιλαμβάνει το γένος του παρόντος υποκειμένου, έτσι ώστε να μπορούν να χρησιμοποιηθούν οι σωστές αντωνυμίες. Όταν βλέπουμε ένα νέο υποκείμενο, θέλουμε να ξεχάσουμε το γένος του παλιού υποκειμένου.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Εικόνα 20: Η πύλη της λήθης και η αντίστοιχη εξίσωση.[26]

Το επόμενο βήμα για το δίκτυο μακράς βραχύχρονης μνήμης είναι να αποφασίσει ποιες νέες πληροφορίες πρόκειται να αποθηκεύσει στο κελί. Αυτό έχει δύο μέρη. Αρχικά, ένα σιγμοειδές επίπεδο που ονομάζεται επίπεδο πύλης εισόδου (input gate layer) αποφασίζει ποιες τιμές θα επικαιροποιήσουμε. Στη συνέχεια, ένα επίπεδο υπερβολικής εφαπτομένης (tanh) δημιουργεί ένα διάνυσμα νέων υποψήφιων τιμών, $\tilde{C}_t$, που θα μπορούσαν να προστεθούν στο κελί. Στο επόμενο βήμα, θα συνδυάσουμε αυτά τα δύο για να δημιουργήσουμε μια επικαιροποίηση για το κελί. Στο παράδειγμα του γλωσσικού μοντέλου, θα θέλαμε να προσθέσουμε το γένος του νέου υποκειμένου στο κελί, για να αντικαταστήσουμε το παλιό που ξεχνάμε.



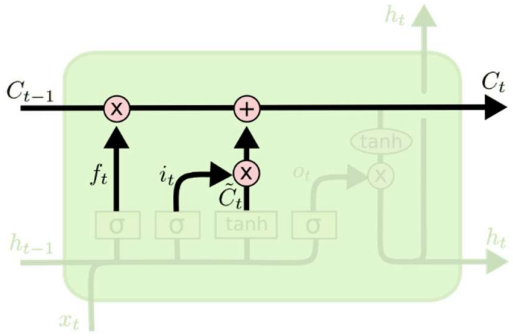
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Εικόνα 21: Η πύλη εισόδου, οι υποψήφιες τιμές και οι αντίστοιχες εξισώσεις.[27]

Τώρα είναι η στιγμή να γίνει η ενημέρωση του παλιού κελιού C_{t-1} σε νέο κελί C_t. Στα προηγούμενα βήματα έχει ήδη αποφασιστεί το τι θα γίνει, μένει μόνο να πραγματοποιηθεί. Πολλαπλασιάζουμε την παλιά κατάσταση με f_t και ξεχνάμε τις πληροφορίες που αποφασίσαμε να ξεχάσουμε νωρίτερα. Έπειτα, προσθέτουμε $i_t * \tilde{C}_t$. Αυτές είναι οι νέες υποψήφιες τιμές, κλιμακωτά με βάση το πώς αποφασίσαμε να επικαιροποιήσουμε κάθε τιμή της κατάστασης.

[26] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .
[27] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .

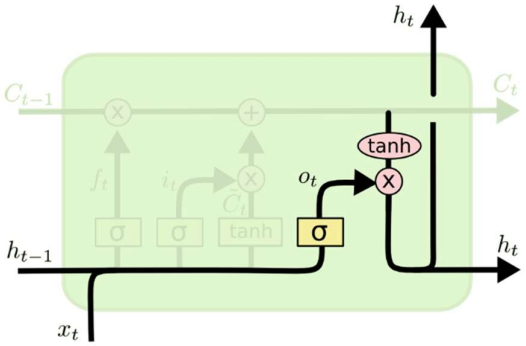
Στην περίπτωση του γλωσσικού μοντέλου, αυτό είναι το σημείο όπου θα ξεχάσουμε τις πληροφορίες για το γένος του παλιού υποκειμένου και θα προσθέσουμε τις νέες πληροφορίες, όπως αποφασίσαμε στα προηγούμενα βήματα.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Εικόνα 22: Επικαιροποίηση των πληροφοριών στην μνήμη του LSTM και η αντίστοιχη εξίσωση.[28]

Τέλος, το δίκτυο μακράς βραχύχρονης μνήμης πρέπει να αποφασίσει τι θα βγάλει ως έξοδο. Η έξοδος αυτή θα βασιστεί στην κατάσταση του κελιού, αλλά θα είναι μία φιλτραρισμένη έκδοση. Πρώτα, ένα σιγμοειδές επίπεδο αποφασίζει ποια κομμάτια του κελιού θα εξαχθούν. Κατόπιν, περνάμε την κατάσταση του κελιού μέσα από υπερβολική εφαπτομένη (για να ωθήσουμε τις τιμές να είναι μεταξύ -1 και 1) και αυτό το πολλαπλασιάζουμε με την έξοδο της σιγμοειδούς πύλης, ώστε να εξάγουμε τα κομμάτια που αποφασίσαμε. Για το παράδειγμα του γλωσσικού μοντέλου, αφού μόλις εντόπισε ένα υποκείμενο, μπορεί να θέλει να εξάγει πληροφορίες σχετικές με ένα ρήμα, στην περίπτωση που αυτό είναι που ακολουθεί. Παραδείγματος χάρη, μπορεί να εξάγει αν ένα υποκείμενο είναι σε ενικό ή πληθυντικό αριθμό, ώστε να καταλάβουμε ποια πρέπει να είναι η κλίση του ρήματος αν αυτό ακολουθεί μετά.



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Εικόνα 23: Η πύλη εξόδου και οι αντίστοιχες εξισώσεις.[29]

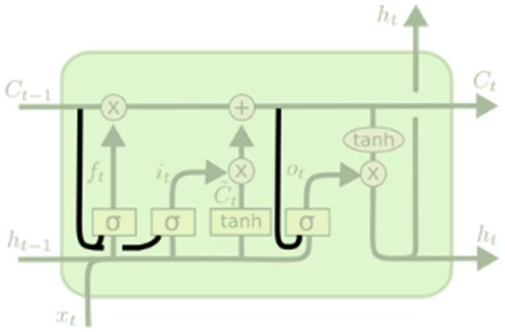
4.1 Παραλλαγές των δικτύων MBM

Ό,τι περιγράψαμε μέχρι τώρα είναι ένα απλό και συνηθισμένο δίκτυο μακράς βραχύχρονης μνήμης. Άλλα δεν είναι όλα τα δίκτυα μακράς βραχύχρονης μνήμης σαν αυτό. Στην πραγματικότητα, φαίνεται ότι σχεδόν κάθε δημοσίευση που σχετίζεται με τα δίκτυα μακράς βραχύχρονης μνήμης χρησιμοποιεί μία λίγο παραλλαγμένη εκδοχή. Οι διαφορές είναι μικρές, αλλά αξίζει να αναφερθούν κάποιες από αυτές.

- Μία δημοφιλής παραλλαγή του δικτύου μακράς βραχύχρονης μνήμης, η οποία εισήχθη από τους Felix Gers και Jurgen Schmidhuber περί το 2000, είναι με την προσθήκη

[28] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .
[29] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .

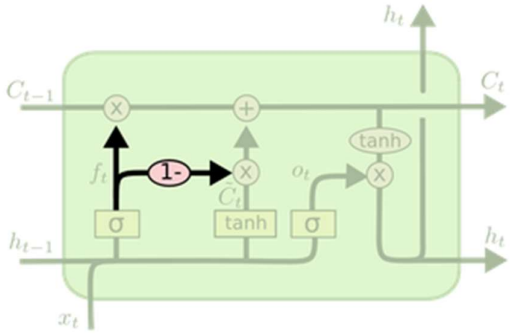
συνδέσεων με τις οποίες τα επίπεδα πύλης «κοιτούν» την κατάσταση του κελιού. Στο παρακάτω διάγραμμα προστίθενται τέτοιες συνδέσεις σε όλες τις πύλες, αλλά πολλές δημοσιεύσεις προσθέτουν κάποιες μόνο συνδέσεις και όχι όλες.



$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$

Εικόνα 24: Διάγραμμα της πρώτης παραλλαγής και οι εξισώσεις των πυλών λήθης, εισόδου και εξόδου.[30]

- Μία άλλη παραλλαγή είναι με τη χρήση της πύλης λήθης και της πύλης εισόδου ως ζευγάρι, αντί να αποφασίζουν ξεχωριστά ποια πληροφορία θα απορριφθεί και ποια θα εισαχθεί στη μνήμη. Με άλλα λόγια, μία πληροφορία θα απορριφθεί όταν πρόκειται να εισαχθεί κάτι άλλο στη θέση της και θα εισαχθούν νέες τιμές όταν θα «ξεχαστεί» κάτι παλιότερο. Η μαθηματική μορφή αυτού είναι η $i_t = 1 - f_t$.

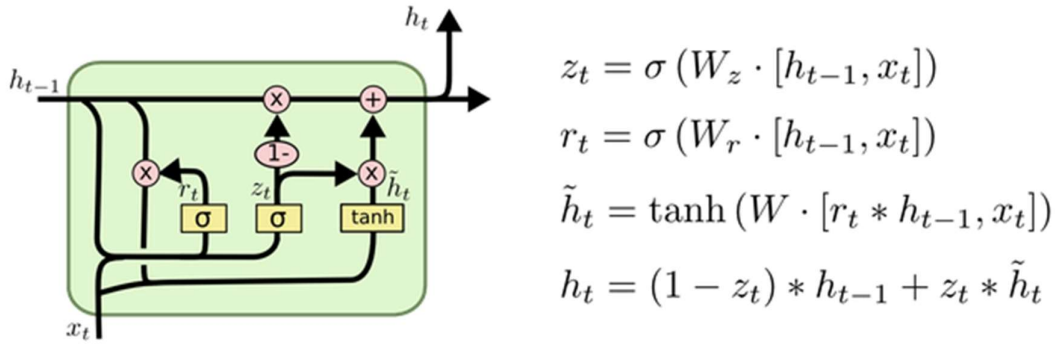


$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Εικόνα 25: Διάγραμμα της δεύτερης παραλλαγής και η εξίσωση της επικαιροποίησης της παλιάς κατάστασης του κελιού σε νέα.[31]

- Μία λίγο πιο έντονη παραλλαγή του δικτύου μακράς βραχύχρονης μνήμης είναι το Gated Recurrent Unit (GRU) που εισήχθη από τον Kyunghyun Cho και άλλους ερευνητές περί το 2014. Συνδυάζει την πύλη λήθης και την πύλη εισόδου σε μία πύλη ενημέρωσης. Ενώνει, επίσης, την κατάσταση του κελιού με την κρυφή κατάσταση και κάνει μερικές ακόμα αλλαγές. Το μοντέλο που προκύπτει είναι πιο απλό από τα κλασικά μοντέλα δικτύων μακράς βραχύχρονης μνήμης και γίνεται όλο και πιο δημοφιλές.

[30] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .
[31] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .



Εικόνα 26: Διάγραμμα του GRU μοντέλου και οι αντίστοιχες εξισώσεις.[32]

Κλείνοντας, αξίζει να αναφέρουμε πως μετά από σύγκριση μεταξύ των δημοφιλών παραλλαγών που έκανε ο Klaus Greff σε συνεργασία με άλλους ερευνητές περί το 2015, αποφάνθηκε ότι τα μοντέλα είναι περίπου ίδια. Επιπλέον, περί το 2015, εξετάστηκαν από τους Rafal Jozefowicz, Wojciech Zaremba και Ilya Sutskever περισσότερες από 10.000 αρχιτεκτονικές επαναλαμβανόμενων νευρωνικών δικτύων και διαπιστώθηκε ότι μερικές δούλευαν καλύτερα από τα δίκτυα μακράς βραχύχρονης μνήμης σε ορισμένες εργασίες.

[32] Πηγή: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ .

ΚΕΦΑΛΑΙΟ 5

ΑΝΑΓΝΩΡΙΣΗ ΤΗΣ ΣΗΜΑΣΙΑΣ ΣΥΜΒΟΛΩΝ ΜΕ ΧΡΗΣΗ ΕΝΔ

Στο κεφάλαιο αυτό, θα γίνει αναφορά στην υλοποίηση προγράμματος σχετικού με την ταξινόμηση δεδομένων εισόδου. Ειδικότερα, για τον σκοπό της παρούσας διπλωματικής εργασίας αναπτύχθηκαν δύο κώδικες. Στόχος είναι με δεδομένα εισόδου μαθηματικά κείμενα TeX (δηλαδή αρχεία γραμμένα με χρήση του λογισμικού LaTeX), ή με άλλα λόγια χαρακτήρες και σημεία στίξης, να κατασκευάσουμε ένα επαναλαμβανόμενο νευρωνικό δίκτυο και στη συνέχεια ένα δίκτυο μακράς βραχύχρονης μνήμης, τα οποία θα εκπαιδευτούν σωστά, με βάση τις μεθόδους που αναλύθηκαν στα προηγούμενα κεφάλαια, ώστε να τα ταξινομούν σε τρεις κατηγορίες: τελεία, υποδιαστολή και λοιπούς χαρακτήρες. Το ζητούμενό μας είναι αυτό, καθώς στα μαθηματικά κείμενα το σύμβολο της τελείας είναι το ίδιο με το σύμβολο της υποδιαστολής (.). Αυτό αποτελεί πρόβλημα, καθώς στον κώδικα Braille, τον οποίο διαβάζουν τα άτομα με προβλήματα όρασης, τα σύμβολα για την τελεία και την υποδιαστολή είναι διαφορετικά. Έτσι, δεν είναι δυνατή η σωστή μεταφορά ενός μαθηματικού κειμένου με τελείες και υποδιαστολές στον κώδικα αυτόν. Με την υλοποίηση του ακόλουθου προγράμματος, λοιπόν, με οποιαδήποτε εισαγωγή κειμένου, το δίκτυο θα είναι σε θέση να εξάγει το επιθυμητό αποτέλεσμα ως έξοδο. Στην περίπτωσή μας, δηλαδή, να ξεχωρίζει την τελεία από την υποδιαστολή.

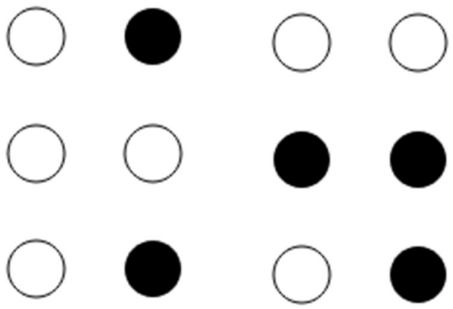
5.1 Υλοποίηση του κώδικα

Για τη δημιουργία του κώδικα, χρησιμοποιήθηκε το προγραμματιστικό περιβάλλον του Google Colaboratory και η γλώσσα προγραμματισμού Python. Εξαιρετικά σημαντικό ρόλο έπαιξε το ανοιχτού κώδικα framework μηχανικής μάθησης PyTorch, το οποίο βασίζεται στη βιβλιοθήκη λογισμικού ανοιχτού κώδικα Torch. Το PyTorch, γενικά, χρησιμοποιείται για εφαρμογές όπως η μηχανική όραση και η επεξεργασία φυσικής γλώσσας (natural language processing ή NLP). Ως επί το πλείστον, αναπτύσσεται από το εργαστήριο τεχνητής νοημοσύνης Meta AI. Επίσης, εγκαταστάθηκαν βιβλιοθήκες, όπως η Unidecode. Η Unidecode παρέχει τη συνάρτηση unidecode(), η οποία λαμβάνει ως είσοδο δεδομένα σε μορφή Unicode και προσπαθεί να τα αναπαραστήσει σε χαρακτήρες ASCII, όπου οι συμβιβασμοί που γίνονται κατά τη χαρτογράφηση μεταξύ των δύο συνόλων χαρακτήρων επιλέγονται να είναι κοντά σε αυτό που θα επέλεγε ένας άνθρωπος με πληκτρολόγιο στη γλώσσα των ΗΠΑ.

Σε αυτό το σημείο, αξίζει να γίνει αναφορά στους δυαδικούς κώδικες, με τους οποίους αναπαρίσταται κάποιο κείμενο ως μία ακολουθία από δυαδικά ψηφία «0» και «1». Οι δυαδικοί κώδικες σταθερού πλάτους χρησιμοποιούν έναν καθορισμένο αριθμό bits για να αναπαραστήσουν κάθε χαρακτήρα στο κείμενο, ενώ στους δυαδικούς κώδικες μεταβλητού πλάτους, ο αριθμός των bits μπορεί να διαφέρει από χαρακτήρα σε χαρακτήρα. Οι δυαδικοί κώδικες χρησιμοποιούνται για την ανάγνωση της γλώσσας του υπολογιστή. Ο ευρέως διαδεδομένος κώδικας ASCII αρχικά ορίστηκε ως ένα σύνολο χαρακτήρων 7-bit. Επιπλέον, με σκοπό να πληροί κάποιες προδιαγραφές, ο κώδικας ASCII επεκτάθηκε στα 8 bits, με την προσθήκη άλλων 128 χαρακτήρων, όπως HP Roman, ISO/IEC 8859, Mac OS Roman και Windows-1252. Ακόμη, το Universal Coded Character Set-2 (UCS-2) αποτελεί μία λίγο ξεπερασμένη κωδικοποίηση με 16 bits ικανή να αναπαραστήσει το βασικό πολύγλωσσο πλάνο της Unicode. Επιπρόσθετα, το Unicode Transformation Format των 32 bits (UTF-32) αποτελεί μία αναπαράσταση της Unicode με 4 bytes για κάθε χαρακτήρα. Όσον αφορά στους δυαδικούς κώδικες μεταβλητού πλάτους, το UTF-8 κωδικοποιεί χαρακτήρες με έναν τρόπο πιο συμβατό με τον κώδικα ASCII, αλλά μπορεί, επίσης, να κωδικοποιήσει όλους τους χαρακτήρες της Unicode με ακολουθίες οι οποίες έχουν έως και 4 bytes των 8 bits. Τέλος, το UTF-16 επεκτείνει το UCS-2 ώστε να καλύπτει ολόκληρη τη Unicode με ακολουθίες αποτελούμενες από ένα ή δύο στοιχεία των 16 bits.

Οι χαρακτήρες του κώδικα Braille αναπαρίστανται χρησιμοποιώντας έξι θέσεις κουκίδας, τοποθετημένες σε ένα ορθογώνιο παραλληλόγραμμο. Κάθε θέση μπορεί να περιέχει μία

υπερυψωμένη κουκίδα ή όχι, επομένως ο κώδικας Braille μπορεί να θεωρηθεί ότι είναι ένας δυαδικός κώδικας των 6 bits.



Εικόνα 27: Στην αριστερή εικόνα απεικονίζεται το σύμβολο της υποδιαστολής στον κώδικα Braille, ενώ στη δεξιά εικόνα απεικονίζεται το σύμβολο της τελείας στον κώδικα Braille.[33]

Συγκεκριμένα, ο κώδικας Braille αποτελεί ένα απτικό σύστημα γραφής το οποίο χρησιμοποιείται από άτομα με προβλήματα όρασης. Μπορεί να διαβαστεί είτε σε ανάγλυφο χαρτί είτε χρησιμοποιώντας ανανεώσιμες οθόνες Braille οι οποίες συνδέονται με υπολογιστές και συσκευές «έξυπνων» κινητών. Ο κώδικας πήρε την ονομασία του από το Γάλλο εφευρέτη του και εκπαιδευτικό Louis Braille, ο οποίος έχασε την όρασή του μετά από κάποιο ατύχημα στην παιδική του ηλικία.

Στην παρούσα διπλωματική εργασία, όπως αναφέρθηκε παραπάνω, το πρόγραμμα που θα υλοποιηθεί δέχεται ως δεδομένα εισόδου μαθηματικά κείμενα. Όσον αφορά στα μαθηματικά, έχει αναπτυχθεί ο κώδικας Nemeth Braille. Ειδικότερα, ο κώδικας Nemeth Braille είναι ένας κώδικας Braille για την κωδικοποίηση μαθηματικής και επιστημονικής σημειογραφίας γραμμικά με χρήση των ορθογωνίων παραλληλογράμμων των 6 κουκίδων. Ο κώδικας αυτός αναπτύχθηκε από τον Abraham Nemeth και γράφτηκε για πρώτη φορά το 1952. Επανεξετάστηκε και διορθώθηκε το 1956, 1965 και 1972 και το 1992 ενσωματώθηκε στον Ενοποιημένο Αγγλικό Braille (Unified English Braille) κώδικα.

Είναι γνωστό ότι η συντριπτική πλειοψηφία των επιστημονικών εγγράφων διεθνώς γράφεται στο TeX και στα παράγωγά του (LaTeX, XeLaTeX, κτλ.). Το ίδιο συμβαίνει και με τα δεδομένα εισόδου του προγράμματος αυτής της εργασίας. Το TeX αποτελεί ένα σύστημα στοιχειοθεσίας, μία τεχνολογία για παραγωγή υψηλής ποιότητας μαθηματικών τυπωμένων εγγράφων, βιβλίων, κτλ., το οποίο σχεδιάστηκε από τον Αμερικανό επιστήμονα πληροφορικής και ομότιμο καθηγητή Donald Knuth και δημοσιεύτηκε το 1978. Το 2017 αναπτύχθηκε πρόγραμμα από τους καθηγητές του Πανεπιστημίου Αιγαίου Ανδρέα Παπασαλούρο και Αντώνιο Τσολομύτη, το οποίο κάνει μετατροπή βιβλίων με Μαθηματικά γραμμένα στο TeX σε Braille. Ωστόσο, υπάρχει η ανάγκη της επίλυσης του ζητήματος του διαχωρισμού της τελείας από την υποδιαστολή, το οποίο ευελπιστεί να λύσει το ακόλουθο πρόγραμμα.

5.2 Μεθοδολογία υλοποίησης

Η ενότητα αυτή ασχολείται αναλυτικά με τη μεθοδολογία ως προς το τεχνικό κομμάτι, η οποία ακολουθήθηκε για την υλοποίηση των κωδίκων. Αναπτύχθηκαν και εκπαιδεύτηκαν ένα αμφίδρομο και ένα απλό (μονόδρομο) επαναλαμβανόμενο νευρωνικό δίκτυο επιπέδου-χαρακτήρα, με στόχο την ταξινόμηση χαρακτήρων σε τρεις κατηγορίες, τελεία, υποδιαστολή και λοιπούς χαρακτήρες. Ένα επαναλαμβανόμενο νευρωνικό δίκτυο αποτελεί ένα δίκτυο το οποίο διατηρεί κάποιου είδους κατάσταση. Ένα επαναλαμβανόμενο νευρωνικό δίκτυο επιπέδου-χαρακτήρα διαβάζει λέξεις ως ακολουθίες χαρακτήρων και εξάγει μία πρόβλεψη και

[33] Πηγή: https://en.wikipedia.org/wiki/Braille

την κρυφή κατάσταση του δικτύου σε κάθε βήμα, τροφοδοτώντας την προηγούμενη κρυφή κατάσταση του δικτύου σε κάθε επόμενο βήμα. Έτσι, η πληροφορία διαδίδεται, όσο το δίκτυο περνά από τα στοιχεία της ακολουθίας. Η τελευταία πρόβλεψη που εξάγεται θεωρείται ως η έξοδος του δικτύου, δηλαδή στην περίπτωσή μας σε ποια κατηγορία ανήκει ο κάθε χαρακτήρας. Τα ακολουθιακά μοντέλα αποτελούν μοντέλα στα οποία υπάρχει κάποιου είδους εξάρτηση μέσα στο χρόνο μεταξύ των εισόδων.

Όσον αφορά στους κώδικες των δικτύων μακράς βραχύχρονης μνήμης, ξανά αναπτύχθηκαν και εκπαιδεύτηκαν ένα αμφίδρομο και ένα απλό (μονόδρομο) δίκτυο μακράς βραχύχρονης μνήμης. Στην περίπτωση αυτή, για κάθε στοιχείο της ακολουθίας, υπάρχει μία αντίστοιχη κρυφή κατάσταση h_t, η οποία, θεωρητικά, μπορεί να περιέχει πληροφορίες από τυχαία σημεία νωρίτερα στην ακολουθία. Το κατά πόσο, όντως, βοήθησαν στην πράξη, ώστε να υπάρξει μεγαλύτερη αποδοτικότητα από τα επαναλαμβανόμενα νευρωνικά δίκτυα, θα αναλυθεί σε παρακάτω ενότητα, έπειτα από την παρατήρηση των γραφικών παραστάσεων και των ανάλυση των αποτελεσμάτων εξόδου.

Το προγραμματιστικό κομμάτι της εργασίας, όπως αναφέρθηκε παραπάνω, αποτελείται από κώδικες ταξινόμησης για την ταξινόμηση των χαρακτήρων των μαθηματικών κειμένων σε τρεις κατηγορίες, τελεία, υποδιαστολή και λοιπούς χαρακτήρες.

1. Κώδικας «bidirectional_rnn»: Αφορά ένα αμφίδρομο επαναλαμβανόμενο νευρωνικό δίκτυο. Το σύνολο των δεδομένων εκπαίδευσης περιλαμβάνει 156 παραδείγματα μαθηματικών κειμένων γραμμένα το ένα κάτω από το άλλο, δηλαδή προτάσεις που συναντώνται σε συγγράμματα μαθηματικού περιεχομένου και αποτελούνται από γράμματα του ελληνικού και του λατινικού αλφάβητου, αριθμούς, μαθηματικά σύμβολα, σημεία στίξης, μεταξύ των οποίων, φυσικά, τελείες και υποδιαστολές. Τα κείμενα αυτά είναι γραμμένα στο TeX. Πιο συγκεκριμένα για τα δεδομένα εκπαίδευσης, εξαιρετικά σημαντική είναι η ένδειξη για τη διαφορετική μορφή της τελείας. Για την ακρίβεια, στη θέση των υποδιαστολών (.) χρησιμοποιήθηκε το σύμβολο της άνω τελείας (·), ενώ στη θέση των τελειών παρέμεινε το σύμβολο της τελείας (.). Με αυτόν τον τρόπο, το δίκτυο κατά την εκπαίδευσή του ξεχωρίζει τα δύο σύμβολα και μαθαίνει να αναγνωρίζει σε ποιο μοτίβο μαθηματικού κειμένου αντιστοιχεί το σύμβολο της υποδιαστολής και σε ποιο το σύμβολο της τελείας. Ακόμη, αξίζει να αναφερθεί ότι μαθηματικές εκφράσεις, οι οποίες σύμφωνα με το σύστημα LaTeX αρχίζουν και τελειώνουν με το σύμβολο του δολαρίου ($), που καταλήγουν σε τελεία γράφτηκαν με δύο τρόπους: (α) πρώτα το σύμβολο του δολαρίου και έπειτα το σύμβολο της τελείας και (β) το αντίθετο. Έτσι, το δίκτυο καθίσταται πιο ευέλικτο στην αναγνώριση διαφορετικών διατυπώσεων. Τα δεδομένα επικύρωσης αποτελούνται από 30 προτάσεις μαθηματικών κειμένων, πάνω στις οποίες δεν έχει εκπαιδευτεί το δίκτυο.
2. Κώδικας «unidirectional_rnn»: Αφορά ένα απλό (μονόδρομο) επαναλαμβανόμενο νευρωνικό δίκτυο. Το σύνολο των δεδομένων εκπαίδευσης περιλαμβάνει τα ίδια παραδείγματα με τον πρώτο κώδικα. Όμοια, τα δεδομένα επικύρωσης αποτελούνται από τις ίδιες προτάσεις με τον πρώτο κώδικα.
3. Κώδικας «bidirectional_lstm»: Αφορά ένα αμφίδρομο δίκτυο μακράς βραχύχρονης μνήμης. Το σύνολο των δεδομένων εκπαίδευσης περιλαμβάνει, ξανά, τα ίδια παραδείγματα. Όμοια, τα δεδομένα επικύρωσης αποτελούνται από τις ίδιες προτάσεις.
4. Κώδικας «unidirectional_lstm»: Αφορά ένα απλό (μονόδρομο) δίκτυο μακράς βραχύχρονης μνήμης. Το σύνολο των δεδομένων εκπαίδευσης περιλαμβάνει, ξανά, τα ίδια παραδείγματα. Όμοια, τα δεδομένα επικύρωσης αποτελούνται από τις ίδιες προτάσεις.

Στις παρακάτω εικόνες παρατίθενται δύο παραδείγματα του συνόλου δεδομένων εκπαίδευσης, καθώς και ένα παράδειγμα πρόβλεψης που εξάγει το δίκτυο:

```
$\pi\simeq 3·14159$.

$\pi\simeq 3·14159.$
```

Εικόνα 28: Δύο παραδείγματα του συνόλου δεδομένων εκπαίδευσης.

```
x
=
1
. υποδιαστολή
2
. τελεία
```

Εικόνα 29: Παράδειγμα πρόβλεψης του δικτύου.

Ο αλγόριθμος σχετικά με την διαδικασία εκπαίδευσης και πρόβλεψης του δικτύου στον κώδικα «bidirectional_rnn» είναι ο εξής:

- Εισαγωγή και δήλωση βιβλιοθηκών.
- Εισαγωγή του συνόλου δεδομένων εκπαίδευσης και δήλωση της τοποθεσίας τους στον υπολογιστή.
- Προετοιμασία των δεδομένων.
- Δημιουργία του μοντέλου του αμφίδρομου επαναλαμβανόμενου νευρωνικού δικτύου.
- Αρχικοποίηση μεταβλητών και υπερπαραμέτρων.
- Εκπαίδευση του δικτύου.
- Δημιουργία γραφικών παραστάσεων με σκοπό την εξαγωγή συμπερασμάτων.
- Διαδικασία πρόβλεψης.
- Εισαγωγή από το χρήστη προτάσεων προς πρόβλεψη και αποτελέσματα.

Ουσιαστικά, αυτό που γίνεται στον κώδικα είναι το εξής:

- ✓ Αρχικά εισάγονται και δηλώνονται οι βιβλιοθήκες και τα πακέτα, προκειμένου να μπορέσουμε στη συνέχεια να πραγματοποιήσουμε τις διαδικασίες που είναι απαραίτητες. Έπειτα, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο κάποιον χαρακτήρα και επιστρέφει την κατηγορία στην οποία ανήκει, δηλαδή επιστρέφει «1» αν ο χαρακτήρας είναι υποδιαστολή, «2» αν είναι τελεία και «3» αν είναι οποιοσδήποτε άλλος χαρακτήρας. Ύστερα, αναπτύσσεται συνάρτηση η οποία ανοίγει ένα αρχείο (στην περίπτωσή μας το αρχείο TeX με τα παραδείγματα μαθηματικού κειμένου) για να διαβαστεί, χρησιμοποιώντας κωδικοποίηση UTF-8. Διαβάζοντας χαρακτήρα-χαρακτήρα το περιεχόμενο του αρχείου, δημιουργεί δύο διανύσματα, ένα με τους χαρακτήρες και ένα με τις κατηγορίες στις αντίστοιχες θέσεις. Στην περίπτωση που ο χαρακτήρας είναι άνω τελεία (·), τον μετατρέπει σε τελεία (.). Κλείνει το αρχείο και επιστρέφει τα δύο διανύσματα που δημιούργησε. Ακολούθως, γίνεται προσάρτηση του φακέλου στο drive με το αρχείο εκπαίδευσης και, καλώντας την παραπάνω συνάρτηση, δημιουργεί τα διανύσματα χαρακτήρων και κατηγοριών από το αρχείο TeX των παραδειγμάτων μαθηματικών κειμένων που έχουμε δημιουργήσει και η μορφή του περιγράφτηκε παραπάνω. Μετά, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο μία ακολουθία χαρακτήρων και διαβάζοντάς τη χαρακτήρα-χαρακτήρα τη μετατρέπει σε τανυστή με διαστάσεις <μήκος_ακολουθίας x 128> (επιλέγουμε 128, δηλαδή όσοι είναι οι χαρακτήρες που περιλαμβάνονται στον κώδικα ASCII), έναν πίνακα με one-hot διανύσματα χαρακτήρων. Ένα διάνυσμα one-hot

απαρτίζεται από μηδενικά σε όλες του τις θέσεις, εκτός από αυτή του εν λόγω γράμματος που παίρνει την τιμή 1. Για παράδειγμα, "b" = <0 1 0 0 0 ...>. Επιστρέφει αυτόν τον τανυστή. Επιπλέον, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο μία ακολουθία. Τη μετατρέπει σε τανυστή με διαστάσεις <μήκος_ακολουθίας x 3> (επιλέγουμε 3, δηλαδή όσες είναι οι κατηγορίες), έναν πίνακα με one-hot διανύσματα κατηγοριών και επιστρέφει αυτόν τον τανυστή. Ακόμη, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο έναν πίνακα των μαθηματικών κειμένων με τον οποίο θα ασχοληθεί, μία θέση στον πίνακα και το μέγεθος μίας ακολουθίας. Μετατρέπει την ακολουθία χαρακτήρων του μέρους του πίνακα από τη δοθείσα θέση έως τη δοθείσα θέση συν το μέγεθος της ακολουθίας σε τανυστή, καλώντας τη συνάρτηση που αναφέραμε παραπάνω και επιστρέφει ως έξοδο τον τανυστή αυτόν. Όμοια, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο έναν πίνακα των μαθηματικών κειμένων με τον οποίο θα ασχοληθεί, μία θέση στον πίνακα και το μέγεθος μίας ακολουθίας. Μετατρέπει την ακολουθία των κατηγοριών του μέρους του πίνακα από τη δοθείσα θέση έως τη δοθείσα θέση συν το μέγεθος της ακολουθίας σε τανυστή, χρησιμοποιώντας τη συνάρτηση που αναλύσαμε παραπάνω και επιστρέφει ως έξοδο τον τανυστή αυτόν. Ουσιαστικά, αυτό που κάνουν αυτή και η προηγούμενη συνάρτηση, είναι να διαβάζουν μία ακολουθία χαρακτήρων ή κατηγοριών, αντίστοιχα, και να επιστρέφουν όλες τις υπακολουθίες με ένα ορισμένο μέγεθος στη σειρά. Επίσης, αναπτύσσεται συνάρτηση η οποία με δεδομένο το μέγεθος μίας ακολουθίας, που δίνεται ως είσοδος, επιστρέφει ένα τυχαίο δείγμα τανυστών χαρακτήρων και τους αντίστοιχους τανυστές κατηγοριών. Επιπρόσθετα, χρησιμοποιείται μία λειτουργία της PyTorch για να ελεγχθεί αν η συσκευή στην οποία εκτελούμε το πρόγραμμα διαθέτει CPU ή GPU. Η υλοποίηση του νευρωνικού δικτύου της παρούσας εργασίας δεν απαιτεί GPU, καθώς η εκπαίδευση του δικτύου δεν είναι πολύ περίπλοκη. Αν GPU είναι διαθέσιμη, θέτουμε τη συσκευή μας σε GPU, ενώ αν GPU δεν είναι διαθέσιμη, θέτουμε τη συσκευή μας σε CPU. Η συσκευή θα χρησιμοποιηθεί στο πρόγραμμα παρακάτω.

- ✓ Σε αυτό το σημείο, κατασκευάζεται το μοντέλο του νευρωνικού συστήματος. Για να γίνει αυτό, ορίζεται μία κλάση η οποία «κληρονομεί» τη βασική κλάση της PyTorch για όλα τα δομοστοιχεία νευρωνικού δικτύου. Γίνεται ανάθεση τιμών στις παραμέτρους της διάστασης του κρυφού επιπέδου και του αριθμού των επιπέδων, αντίστοιχα, και ορίζονται τα επίπεδα του μοντέλου. Για το μοντέλο αυτό, θα χρειαστεί 1 επίπεδο επαναλαμβανόμενου νευρωνικού δικτύου (όπου με την εντολή bidirectional=True, το επαναλαμβανόμενο νευρωνικό δίκτυο ορίζεται ως αμφίδρομο) και ένα πλήρως συνδεδεμένο επίπεδο (όπου διπλασιάζεται η διάσταση του κρυφού επιπέδου, αφού έχουμε αμφίδρομο δίκτυο). Το πλήρως συνδεδεμένο επίπεδο είναι υπεύθυνο για τη μετατροπή της εξόδου του επαναλαμβανόμενου νευρωνικού δικτύου στο επιθυμητό σχήμα εξόδου. Επίσης, ορίζεται η συνάρτηση του forward pass. Στη συνάρτηση αυτή, αρχικοποιείται η κατάσταση του κρυφού επιπέδου για την πρώτη είσοδο καλώντας τη συνάρτηση που ορίζεται αμέσως παρακάτω, η οποία δημιουργεί έναν τανυστή με μηδενικά στο σχήμα των κρυφών επιπέδων, ο οποίος χρησιμοποιείται ως η πρώτη κρυφή κατάσταση στο forward pass. Έπειτα, περνάμε τα δεδομένα εισόδου και την κατάσταση του κρυφού επιπέδου, μέσα στο επίπεδο του επαναλαμβανόμενου νευρωνικού δικτύου. Στη συνέχεια, γίνεται αλλαγή του σχήματος των δεδομένων εξόδου έτσι ώστε να είναι κατάλληλο για το πλήρως συνδεδεμένο δίκτυο και περνάμε αυτά τα δεδομένα στο πλήρως συνδεδεμένο επίπεδο. Η συνάρτηση του forward pass βγάζει ως έξοδο τα δεδομένα εξόδου και την κρυφή κατάσταση του επαναλαμβανόμενου νευρωνικού δικτύου.
- ✓ Στο σημείο αυτό γίνεται η αρχικοποίηση και η εκπαίδευση του δικτύου. Ορίζονται το μέγεθος των χαρακτήρων σε 128 (δηλαδή ο τανυστής ενός χαρακτήρα έχει μέγεθος όσοι είναι οι χαρακτήρες που περιλαμβάνονται στον κώδικα ASCII), το μέγεθος των κατηγοριών σε 3, το μέγεθος της ακολουθίας που διαβάζει το πρόγραμμα κάθε φορά και ο αριθμός των κρυφών επιπέδων. Με τις τιμές των υπερπαραμέτρων αυτών

αρχικοποιείται το μοντέλο, το οποίο εισάγεται στη CPU ή GPU, αναλόγως, ώστε να μας δώσει αποτελέσματα. Ακολούθως, ορίζεται ο αριθμός των επαναλήψεων της τροφοδότησης παραδειγμάτων στο δίκτυο και του ρυθμού εκμάθησης και επιλέγεται ως συνάρτηση απώλειας η διασταυρούμενη εντροπία, καθώς αποτελεί κατάλληλη επιλογή για εργασίες ταξινόμησης. Ως βελτιστοποιητής ορίζεται ο adam. Ύστερα, εκτελείται η εκπαίδευση του δικτύου για τον αριθμό επαναλήψεων που ορίστηκε. Πρώτα, μηδενίζονται οι κλίσεις της προηγούμενης επανάληψης. Μετά, με κλήση παραπάνω συνάρτησης επιλέγεται ένα τυχαίο δείγμα ακολουθίας χαρακτήρων από τα παραδείγματα και της αντίστοιχης ακολουθίας των κατηγοριών τους. Η ακολουθία των χαρακτήρων εισάγεται στο μοντέλο και αυτό μας δίνει αποτελέσματα. Έτσι, υπολογίζεται το σφάλμα συγκρίνοντας τις προβλέψεις του μοντέλου με τις πραγματικές κατηγορίες των χαρακτήρων. Τελείται οπισθοδιάδοση, υπολογίζονται οι κλίσεις και ενημερώνονται τα βάρη αντιστοίχως. Τέλος, δίνονται οι εντολές για να τυπώνεται στην οθόνη η εκάστοτε επανάληψη και το αντίστοιχο σφάλμα, καθώς, και να σχηματίζεται η γραφική παράσταση του σφάλματος εκπαίδευσης.

- ✓ Τέλος, γίνεται η δοκιμή του δικτύου. Αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο μία πρόταση και τη μετατρέπει σε one-hot κωδικοποίηση, ώστε να μπορεί να εισαχθεί στο μοντέλο. Το μοντέλο δίνει την πρόβλεψή του. Η πρόβλεψη αυτή τυπώνεται στην οθόνη ως εξής: τυπώνεται ο ένας χαρακτήρας κάτω από τον άλλο και δίπλα από κάθε χαρακτήρα, αν αυτός είναι τελεία, τυπώνεται «τελεία», αν είναι υποδιαστολή, τυπώνεται «υποδιαστολή» και αν είναι οτιδήποτε άλλο, τυπώνεται χωρίς κάτι δίπλα.

Όμοια, ο αλγόριθμος σχετικά με τη διαδικασία εκπαίδευσης και πρόβλεψης του δικτύου στον κώδικα «unidirectional_rnn» περιλαμβάνει τα ίδια βήματα. Οι μόνες αλλαγές είναι μέσα στο μοντέλο, όπου δε χρειάζεται ο διπλασιασμός των επιπέδων του δικτύου, αφού είναι μονόδρομο.

Ο αλγόριθμος σχετικά με την διαδικασία εκπαίδευσης και πρόβλεψης του δικτύου στον κώδικα «bidirectional_lstm» είναι ο εξής:

- Εισαγωγή και δήλωση βιβλιοθηκών.
- Εισαγωγή του συνόλου δεδομένων εκπαίδευσης και δήλωση της τοποθεσίας τους στον υπολογιστή.
- Προετοιμασία των δεδομένων.
- Δημιουργία του μοντέλου του δικτύου μακράς βραχύχρονης μνήμης.
- Εφαρμογή τεχνικών εξομάλυνσης.
- Αρχικοποίηση μεταβλητών και υπερπαραμέτρων.
- Εκπαίδευση του δικτύου.
- Δημιουργία γραφικών παραστάσεων με σκοπό την εξαγωγή συμπερασμάτων.
- Διαδικασία πρόβλεψης.
- Εισαγωγή από το χρήστη προτάσεων προς πρόβλεψη και αποτελέσματα.

Όμοια, ο αλγόριθμος σχετικά με τη διαδικασία εκπαίδευσης και πρόβλεψης του δικτύου στον κώδικα «unidirectional_lstm» περιλαμβάνει τα ίδια βήματα. Οι μόνες αλλαγές είναι μέσα στο μοντέλο, όπου δε χρειάζεται ο διπλασιασμός των επιπέδων του δικτύου, αφού είναι μονόδρομο.

Ουσιαστικά, οι μόνες διαφορές ανάμεσα στους αλγορίθμους των κωδίκων των επαναλαμβανόμενων δικτύων και των δικτύων μακράς βραχύχρονης μνήμης είναι, φυσικά, η αλλαγή από μοντέλο επαναλαμβανόμενου νευρωνικού δικτύου σε δίκτυο μακράς βραχύχρονης μνήμης με την εύκολη τροποποίηση κάποιων εντολών και η παρουσία τεχνικών εξομάλυνσης, dropout και L2 εξομάλυνση, στους κώδικες των δικτύων μακράς βραχύχρονης μνήμης.

Όσον αφορά στην επιλογή του αριθμού των επαναλήψεων τροφοδότησης παραδειγμάτων στο δίκτυο, της τιμής του ρυθμού εκμάθησης, της μείωσης βάρους και, εν γένει, της αρχιτεκτονικής του δικτύου, καθοριστικός ήταν ο πειραματισμός με δοκιμές διαφορετικών αριθμών και η προσπάθεια εξαγωγής κάποιου συμπεράσματος για κάθε έννοια. Ειδικότερα, ο αριθμός των επαναλήψεων επιλέχθηκε έτσι ώστε να είναι μεγάλος και, επομένως, η τιμή της ακρίβειας να είναι υψηλή, γεγονός που υποδηλώνει ότι η εκπαίδευση του δικτύου έγινε επιτυχώς. Στο αμφίδρομο και μονόδρομο επαναλαμβανόμενο δίκτυο παρατηρήθηκε ότι με την αρχική ανάθεση τιμής στις 25000, ο αλγόριθμος είναι πολύ αποδοτικός. Όμοια, στα δίκτυα μακράς βραχύχρονης μνήμης, παρατηρήθηκε ότι με την αρχική ανάθεση τιμής στις 25000, ο αλγόριθμος είναι πολύ αποδοτικός. Στις δοκιμές, όμως, με μεγαλύτερο αριθμό επαναλήψεων παρατηρήθηκε ότι μειώνεται η ακρίβεια του δικτύου. Αυτό εξηγείται από το ότι αν τεθεί μεγαλύτερος αριθμός από τον κατάλληλο, τότε θα μπορούσαμε να πούμε ότι το δίκτυο απομνημονεύει μόνο τα δεδομένα εκπαίδευσης και υπάρχει κίνδυνος υπερπροσαρμογής. Όσον αφορά στον ρυθμό εκμάθησης, επιλέχθηκε η τιμή 0.001 και για όλους τους κώδικες. Ο λόγος είναι ότι παρατηρήθηκε πως όταν ο ρυθμός εκμάθησης είναι χαμηλότερος, η εκπαίδευση επιβραδύνεται, αφού οι ενημερώσεις των βαρών είναι πολύ μικρές. Στον αντίποδα, όταν ο ρυθμός εκμάθησης είναι υψηλότερος, η γραφική παράσταση της συνάρτησης απώλειας παρουσιάζει απόκλιση. Σχετικά με την παράμετρο της μείωσης του βάρους, δηλαδή την παράμετρο εξομάλυνσης λ, όπως αναφέραμε και στο θεωρητικό μέρος της εργασίας, η επιλογή τιμής συνήθως γίνεται με λογαριθμική κλίμακα μεταξύ του 0 και του 0.1, όπως 0.1, 0.001, 0.0001 κτλ. Έτσι, επιλέγοντας αρχικά 0.0001, διαπιστώθηκε ότι το δίκτυο ήταν αρκετά αποδοτικό. Προτείνεται, όμως, στο μέλλον να δοκιμαστεί ο κώδικας και με άλλες τιμές της μείωσης του βάρους.

Όσον αφορά στις παραμέτρους του μοντέλου, η παράμετρος input_size αφορά στον αριθμό των αναμενόμενων στοιχείων της εισόδου x. Στην περίπτωσή μας είναι 128, όσο είναι το one-hot μέγεθος των χαρακτήρων, αφού ο κώδικας ASCII περιλαμβάνει ορισμούς για 128 χαρακτήρες. Η παράμετρος output_size αφορά το μέγεθος της εξόδου του δικτύου και παίρνει την τιμή 3, όσες και οι κατηγορίες στις οποίες θέλουμε να ταξινομήσουμε τους χαρακτήρες των μαθηματικών κειμένων. Η παράμετρος hidden_dim αφορά στον αριθμό των στοιχείων στα κρυφά επίπεδα. Όταν δεν υπάρχει πάρα πολύ μεγάλος όγκος δεδομένων εισόδου, καλό είναι να κυμαίνεται σε μικρές τιμές. Για αυτόν το λόγο όταν επιλέξαμε μεγάλες τιμές το δίκτυο πάθαινε υπερπροσαρμογή και, έτσι, καταλήξαμε στην τιμή 12, με την οποία το δίκτυο ήταν αποδοτικό. Η παράμετρος n_layers αφορά τον αριθμό των επαναλαμβανόμενων επιπέδων και επιλέχθηκε η προεπιλεγμένη τιμή 1. Σχετικά με το μέγεθος της παρτίδας, η ποσότητα των δεδομένων μας δεν είναι πολύ μεγάλη και, γι' αυτό αποφασίσαμε να μην χωρίσουμε τα δεδομένα σε παρτίδες. Έτσι, επιλέχθηκε η προεπιλεγμένη τιμή 1, δηλαδή το δίκτυο εκπαιδεύεται με 1 παράδειγμα τη φορά. Προτείνεται στο μέλλον να γίνουν δοκιμές με μεγαλύτερες τιμές μεγέθους παρτίδας.

Συμπερασματικά, αποδεικνύεται ότι πολλές από τις μεθοδολογίες που αναφέρθηκαν κατά την παρουσίαση του θεωρητικού μέρους της παρούσας εργασίας ήταν εξαιρετικά χρήσιμες για την εκπαίδευση και την εξαγωγή καλών αποτελεσμάτων από το δίκτυο. Αναλυτική περιγραφή των βημάτων της μεθοδολογίας που πραγματοποιήθηκαν στους κώδικες, γίνεται σε επόμενη ενότητα.

5.3 Συμπεράσματα

Σε αυτή την ενότητα, θα γίνει αναφορά στα αποτελέσματα που προέκυψαν από τους κώδικες ταξινόμησης, έπειτα από την εκπαίδευση των δικτύων και τις προβλέψεις που εξήγαγαν με σύνολο δεδομένων επικύρωσης παραδείγματα αρκετά διαφορετικά από αυτά του συνόλου δεδομένων εκπαίδευσης. Επιπλέον, θα γίνει εξαγωγή συμπερασμάτων. Καταρχάς, ιδιαίτερα σημαντικό να σημειωθεί είναι ότι το ζητούμενο από τη γραφική παράσταση είναι οι απώλειες να είναι μικρές, ώστε να θεωρηθεί πως έχει εκπαιδευτεί καλά το δίκτυο. Ακόμη, όταν, για

παράδειγμα, η απώλεια είναι μικρή, ο κώδικας πρέπει να εξάγει σωστά αποτελέσματα, ώστε να μην υπάρχει υπερπροσαρμογή.

Το πρώτο συμπέρασμα που εξηγάγαμε προέκυψε από την υλοποίηση του κώδικα «bidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης 51 παραδειγμάτων. Ο κώδικας αυτός, ενώ είχε πολύ μικρό σφάλμα εκπαίδευσης το οποίο σύγκλινε πολύ γρήγορα στο μηδέν, εντούτοις είχε και μεγάλο σφάλμα επικύρωσης. Επομένως, διαπιστώσαμε ότι το δίκτυο παθαίνει υπερπροσαρμογή και συμπεράναμε ότι ένα τέτοιο δίκτυο χρειάζεται περισσότερα δεδομένα εκπαίδευσης προκειμένου να εκπαιδευτεί σωστά. Για το λόγο αυτό, προσθέσαμε και άλλα παραδείγματα μαθηματικού κειμένου στο σύνολο δεδομένων εκπαίδευσης, φτάνοντας τα 156.

Το δεύτερο συμπέρασμα που εξηγάγαμε προέκυψε από τη σύγκριση μεταξύ των μονόδρομων (απλών) και των αμφίδρομων επαναλαμβανόμενων νευρωνικών δικτύων και δικτύων μακράς βραχύχρονης μνήμης. Συγκεκριμένα, παρατηρώντας τα αποτελέσματα συμπεράναμε ότι και στις δύο αρχιτεκτονικές δικτύων (δηλαδή επαναλαμβανόμενα δίκτυα και δίκτυα μακράς βραχύχρονης μνήμης) πιο αποδοτικά ήταν τα αμφίδρομα δίκτυα. Το γεγονός αυτό ήταν αρκετά αναμενόμενο, καθώς, όπως αναφέρθηκε στο θεωρητικό μέρος της παρούσας εργασίας, τα αμφίδρομα δίκτυα, γενικά, δίνουν καλύτερα αποτελέσματα σε εργασίες ταξινόμησης.

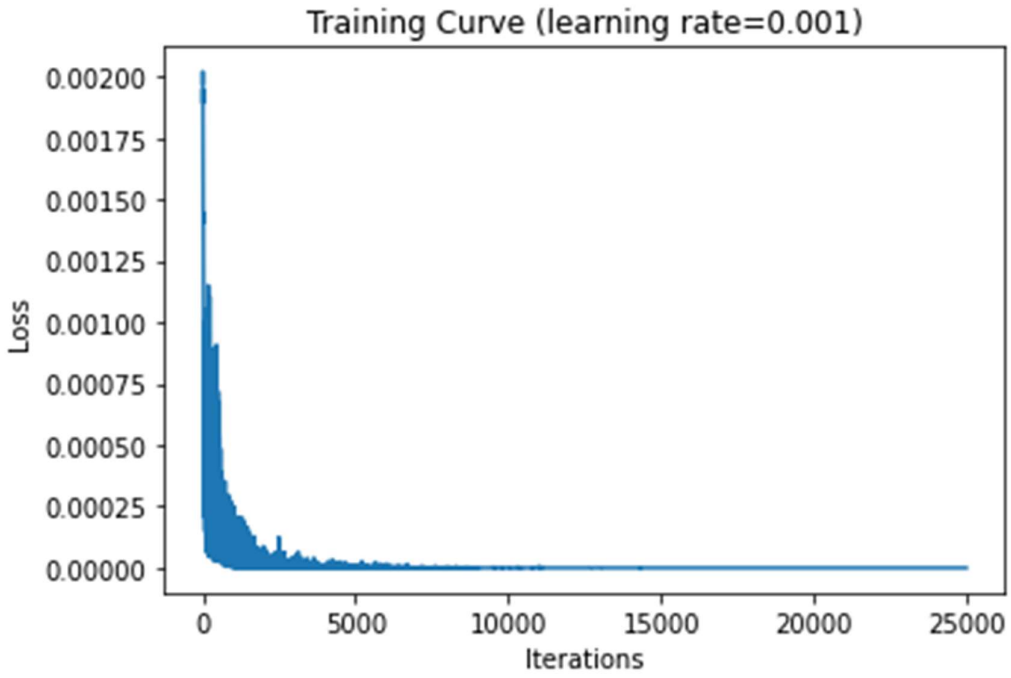
Το τρίτο συμπέρασμα προέκυψε μετά από σύγκριση των επαναλαμβανόμενων δικτύων με τα δίκτυα μακράς βραχύχρονης μνήμης. Ειδικότερα, συμπεράναμε ότι, αν και το αμφίδρομο επαναλαμβανόμενο δίκτυο προέβλεψε με αρκετή ακρίβεια τις κατηγορίες των χαρακτήρων, δεν έφτασε την ακρίβεια του αμφίδρομου δικτύου μακράς βραχύχρονης μνήμης. Όμοια, η ακρίβεια πρόβλεψης του μονόδρομου επαναλαμβανόμενου δικτύου ήταν αρκετά χαμηλότερη από την ακρίβεια του μονόδρομου δικτύου μακράς βραχύχρονης μνήμης. Το γεγονός αυτό συμπίπτει με το θεωρητικό μέρος αυτής της διπλωματικής εργασίας, όπου αναφέρθηκε ότι τα δίκτυα μακράς βραχύχρονης μνήμης, γενικά, δίνουν καλύτερα αποτελέσματα σε εργασίες ταξινόμησης ακολουθιακών δεδομένων.

Τέλος, έγινε σύγκριση μεταξύ του αμφίδρομου δικτύου μακράς βραχύχρονης μνήμης με αριθμό χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) 30 με το ίδιο δίκτυο αλλάζοντας τον αριθμό χαρακτήρων που διαβάζει το δίκτυο κάθε φορά σε 60. Παρατηρώντας ότι το δίκτυο το οποίο διαβάζει 60 χαρακτήρες τη φορά έχει την καλύτερη απόδοση, συμπεράναμε ότι δίκτυα με ακολουθιακά δεδομένα και, γενικά, με σύνολο δεδομένων εκπαίδευσης αριθμού όμοιου με το δίκτυο της παρούσας εργασίας, αποδίδουν καλύτερα όταν δε διαβάζουν μικρό αριθμό των στοιχείων της ακολουθίας τη φορά.

Συνοψίζοντας την ενότητα αυτή, συμπεράναμε ότι το δίκτυο με την καλύτερη απόδοση εξ όσων υλοποιήθηκαν είναι το αμφίδρομο δίκτυο μακράς βραχύχρονης μνήμης, το οποίο διαβάζει 60 χαρακτήρες τη φορά από το σύνολο δεδομένων εισόδου. Πιο συγκεκριμένα, η ακρίβεια πρόβλεψης αυτού του δικτύου είναι 93.33%, ποσοστό ενός πολύ καλά εκπαιδευμένου δικτύου.

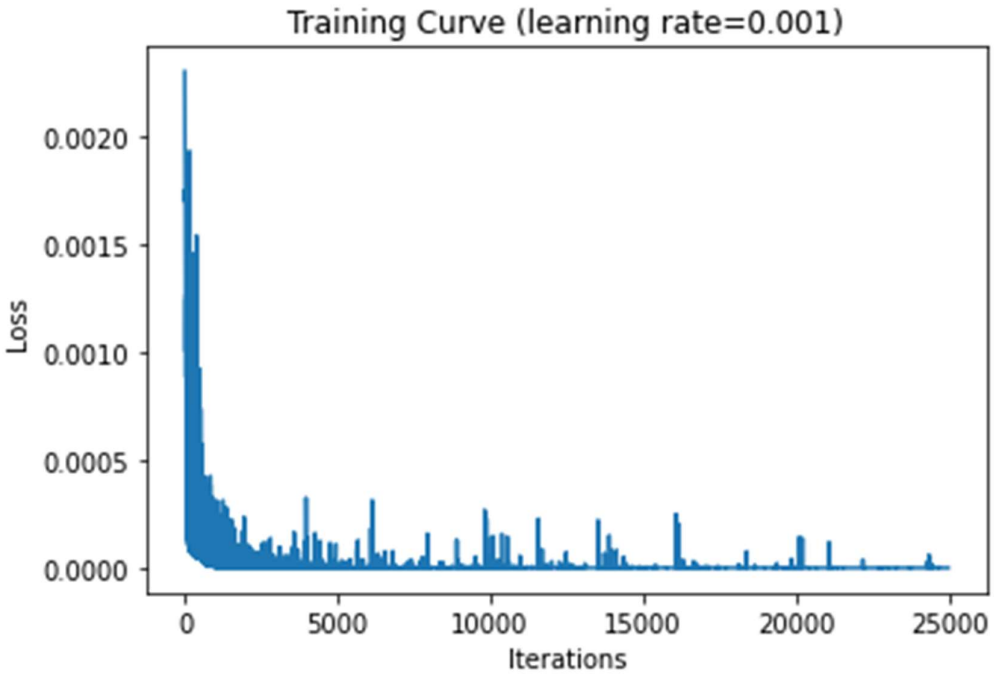
5.4 Δοκιμές που πραγματοποιήθηκαν

Στην ενότητα αυτή θα παρουσιαστούν κάποιες από τις δοκιμές που πραγματοποιήθηκαν, προκειμένου να έχουμε τα συγκεκριμένα αποτελέσματα και να εξάγουμε τα παραπάνω συμπεράσματα. Δημιουργήθηκαν κώδικες και έγιναν αρκετές δοκιμές και συγκρίσεις σε αυτούς, ώστε να καταλήξουμε σε αυτόν με το βέλτιστο αποτέλεσμα.



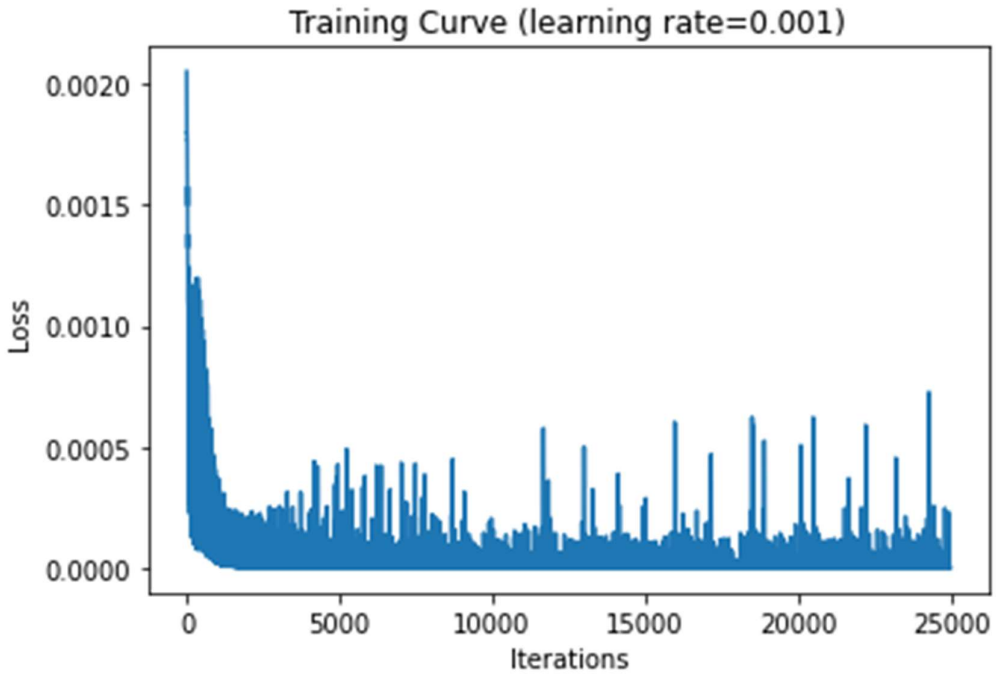
Εικόνα 30: Αποτελέσματα κώδικα «bidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης 51 παραδειγμάτων.

Ο πρώτος κώδικας που δημιουργήθηκε είναι το αμφίδρομο επαναλαμβανόμενο δίκτυο «bidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης 51 παραδείγματα μαθηματικού κειμένου, αριθμό χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) 15 και αριθμό επαναλήψεων 25000. Από τη γραφική παράσταση προκύπτει ότι το σφάλμα εκπαίδευσης είναι αρκετά μικρό. Έτσι, είναι αναμενόμενο το δίκτυο να κάνει ορθές προβλέψεις κατά την επικύρωση (testing). Ωστόσο, παρατηρήσαμε ότι το σφάλμα επικύρωσης ήταν μεγάλο, αφού το δίκτυο έκανε σωστές προβλέψεις με ακρίβεια μόνο 66.67%, ποσοστό μικρό για μία τόσο καλή εκπαίδευση. Επομένως, διαπιστώσαμε, όπως αναφέρθηκε και στην προηγούμενη ενότητα, ότι το δίκτυο παθαίνει υπερπροσαρμογή. Προκειμένου να λυθεί αυτό το πρόβλημα, προσθέσαμε και άλλα παραδείγματα μαθηματικού κειμένου στο σύνολο δεδομένων εκπαίδευσης, φτάνοντας τα 156.



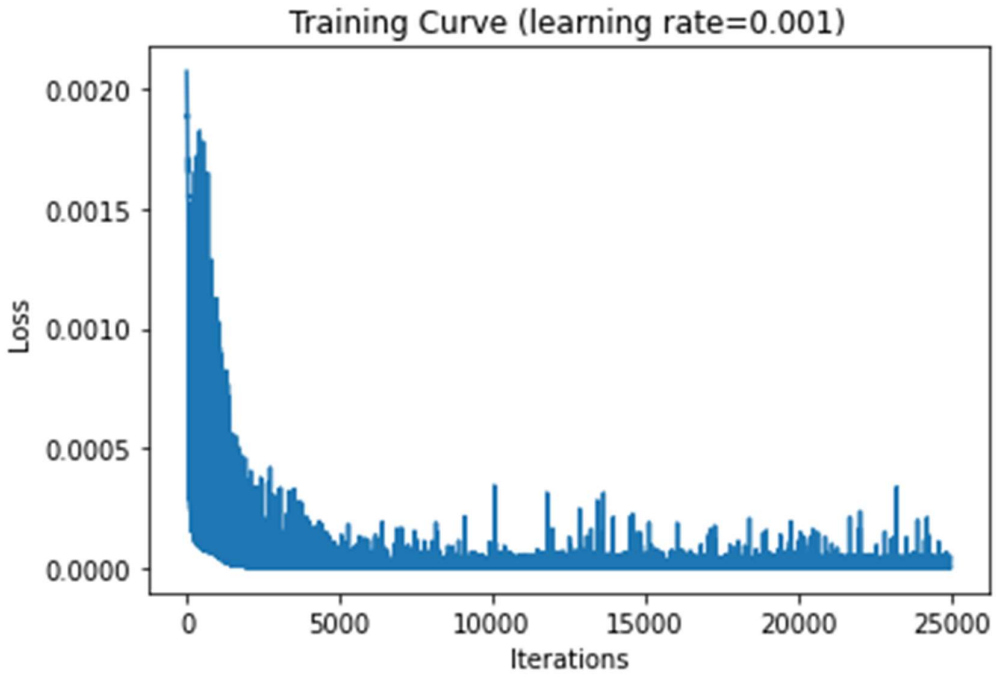
Εικόνα 31: Αποτελέσματα κώδικα «bidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Η παραπάνω εικόνα απεικονίζει τη γραφική παράσταση των αποτελεσμάτων που εξήχθησαν από την εκπαίδευση του δικτύου «bidirectional_rnn» όταν χρησιμοποιήθηκαν ως δεδομένα εκπαίδευσης αυτά τα 156 παραδείγματα. Από τη γραφική παράσταση προκύπτει ότι, καθώς αυξάνονται οι επαναλήψεις, το σφάλμα εκπαίδευσης μειώνεται αρκετά. Το γεγονός αυτό, αυτή τη φορά, αντικατοπτρίζεται και στο σφάλμα επικύρωσης. Συγκεκριμένα, το δίκτυο έκανε σωστές προβλέψεις με ακρίβεια 86.67%, ποσοστό αρκετά μεγαλύτερο από την προηγούμενη δοκιμή, αλλά με ποσοστό λάθους που δεν μπορεί να παραβλεφθεί.



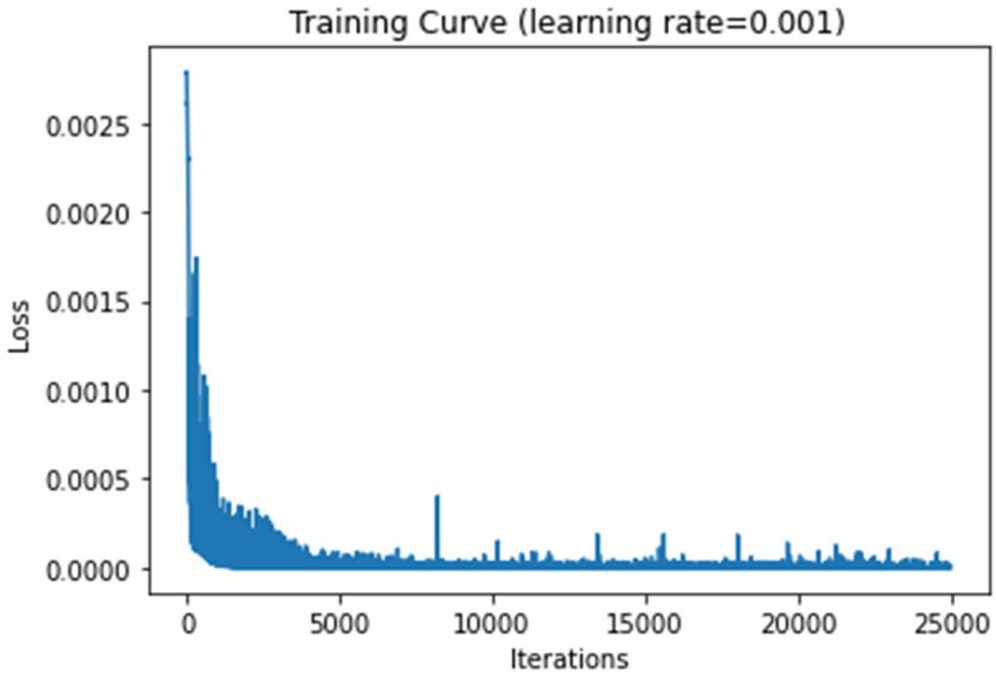
Εικόνα 32: Αποτελέσματα κώδικα «unidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Στη συνέχεια, δημιουργήθηκε το απλό (μονόδρομο) επαναλαμβανόμενο νευρωνικό δίκτυο «unidirectional_rnn» με σύνολο δεδομένων εκπαίδευσης τα ίδια 156 παραδείγματα, αριθμό χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) 15 και αριθμό επαναλήψεων 25000, προκειμένου να συγκριθεί με το αμφίδρομο. Από τη γραφική παράσταση προκύπτει ότι το σφάλμα εκπαίδευσης δε μειώνεται αρκετά. Όσον αφορά στο σφάλμα επικύρωσης, το δίκτυο έκανε σωστές προβλέψεις με ακρίβεια 76.67%, ποσοστό μικρότερο από το αμφίδρομο δίκτυο.



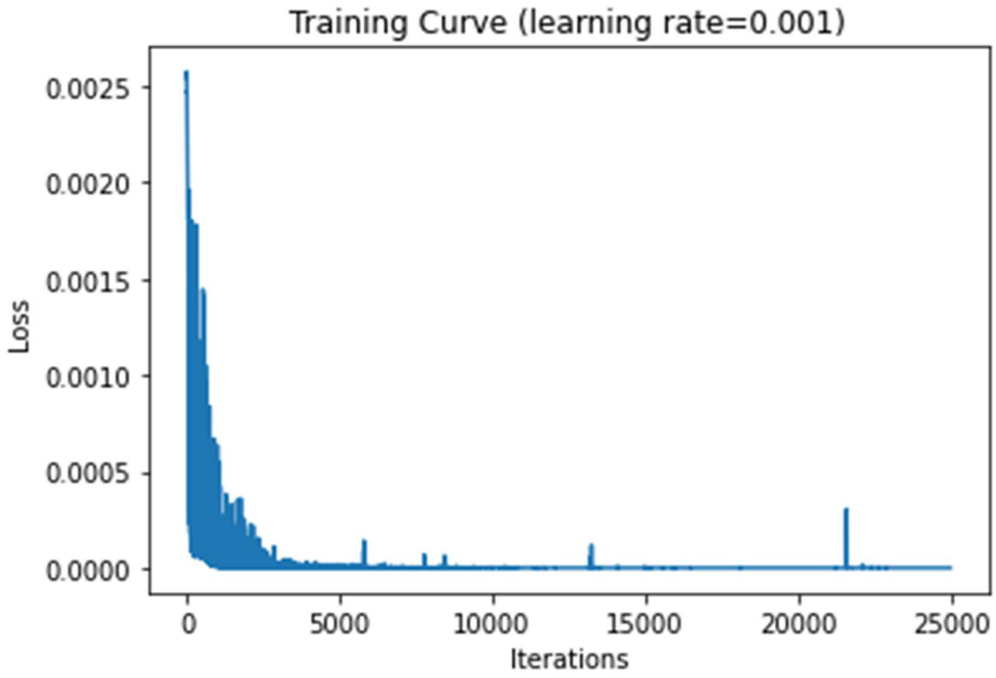
Εικόνα 33: Αποτελέσματα κώδικα «unidirectional_lstm_30» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Έπειτα, δημιουργήθηκαν τα αντίστοιχα δίκτυα, αλλά με κατάλληλες αλλαγές ώστε να γίνουν δίκτυα μακράς βραχύχρονης μνήμης και να συγκριθούν με τα επαναλαμβανόμενα δίκτυα. Πρώτα, δημιουργήθηκε το απλό (μονόδρομο) δίκτυο μακράς βραχύχρονης μνήμης «unidirectional_lstm_30» με σύνολο δεδομένων εκπαίδευσης τα ίδια 156 παραδείγματα. Ο αριθμός χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) αυξήθηκε στους 30, διότι διαπιστώθηκε ότι με μικρότερο αριθμό τα αποτελέσματα είχαν πολύ μεγάλα ποσοστά λάθους. Ο αριθμός επαναλήψεων παρέμεινε στις 25000. Από τη γραφική παράσταση προκύπτει ότι έχει μικρότερο σφάλμα εκπαίδευσης από το απλό επαναλαμβανόμενο δίκτυο, αλλά μεγαλύτερο από το αμφίδρομο. Το ίδιο συμβαίνει και με το σφάλμα επικύρωσης, σύμφωνα με το οποίο το δίκτυο έκανε ορθές προβλέψεις με ακρίβεια 83.33%.



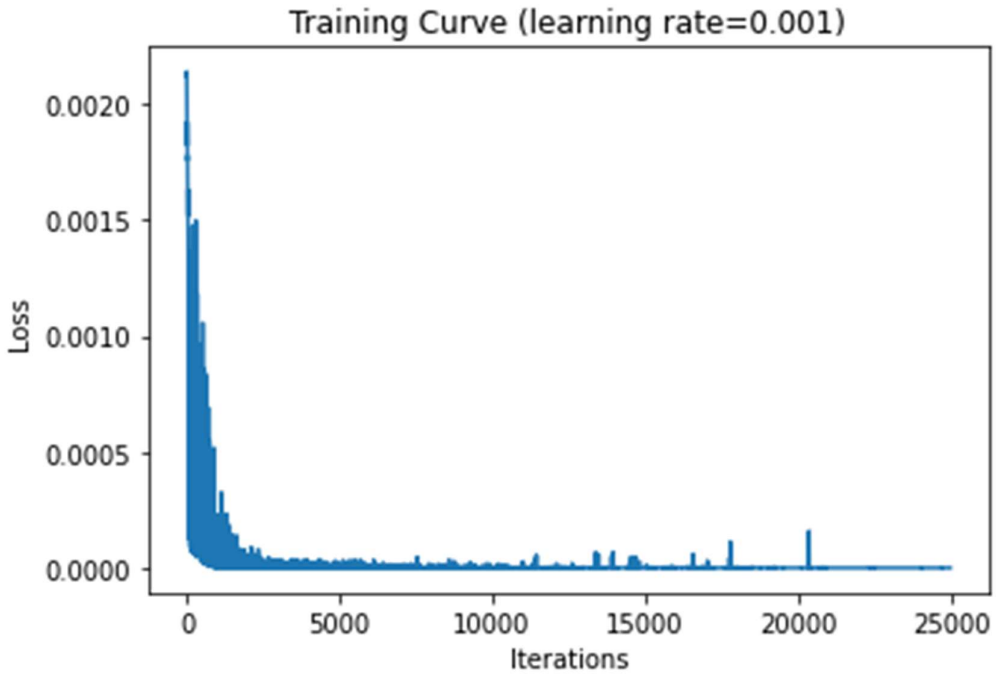
Εικόνα 34: Αποτελέσματα κώδικα «unidirectional_lstm_60» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Ακολούθως, δημιουργήθηκε το απλό (μονόδρομο) δίκτυο μακράς βραχύχρονης μνήμης «unidirectional_lstm_60» με σύνολο δεδομένων εκπαίδευσης τα ίδια 156 παραδείγματα. Ο αριθμός χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) αυξήθηκε στους 60, ώστε να γίνει σύγκριση με το αντίστοιχο δίκτυο των 30. Ο αριθμός επαναλήψεων παρέμεινε στις 25000. Από τη γραφική παράσταση προκύπτει ότι έχει αρκετά μικρότερο σφάλμα εκπαίδευσης από το προηγούμενο δίκτυο, αλλά κινείται σε παρόμοια επίπεδα με το αμφίδρομο των 156 παραδειγμάτων. Το ίδιο συμβαίνει και με το σφάλμα επικύρωσης, σύμφωνα με το οποίο το δίκτυο έκανε ορθές προβλέψεις με ακρίβεια 86.67%.



Εικόνα 35: Αποτελέσματα κώδικα «bidirectional_lstm_30» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Ύστερα, δημιουργήθηκε το αμφίδρομο δίκτυο μακράς βραχύχρονης μνήμης «bidirectional_lstm_30» με σύνολο δεδομένων εκπαίδευσης τα ίδια 156 παραδείγματα, ο αριθμός χαρακτήρων που διαβάζει το δίκτυο κάθε φορά (seq_size) είναι 30 και ο αριθμός επαναλήψεων παρέμεινε στις 25000. Από τη γραφική παράσταση προκύπτει ότι έχει το μικρότερο σφάλμα εκπαίδευσης που έχει προκύψει έως τώρα. Το ίδιο συμβαίνει και με το σφάλμα επικύρωσης, σύμφωνα με το οποίο το δίκτυο έκανε ορθές προβλέψεις με ακρίβεια 90%, ποσοστό αρκετά ικανοποιητικό, ενός αποδοτικού αλγορίθμου.



Εικόνα 36: Αποτελέσματα κώδικα «bidirectional_lstm_60» με σύνολο δεδομένων εκπαίδευσης 156 παραδειγμάτων.

Τέλος, παρατηρώντας ότι το μονόδρομο δίκτυο μακράς βραχύχρονης μνήμης με seq_size=30 ήταν λιγότερο αποδοτικό από το αντίστοιχο δίκτυο με seq_size=60, δημιουργήσαμε το αμφίδρομο δίκτυο μακράς βραχύχρονης μνήμης «bidirectional_lstm_60» με seq_size=60, ώστε να γίνει η σύγκριση με το αντίστοιχο με seq_size=30. Ως δεδομένα εκπαίδευσης χρησιμοποιήθηκαν τα ίδια 156 παραδείγματα και ο αριθμός επαναλήψεων παρέμεινε στις 25000. Από τη γραφική παράσταση προκύπτει ότι έχει ακόμα μικρότερο σφάλμα εκπαίδευσης από τον προηγούμενο κώδικα. Το ίδιο συμβαίνει και με το σφάλμα επικύρωσης, σύμφωνα με το οποίο το δίκτυο έκανε ορθές προβλέψεις με ακρίβεια 93.33%, ποσοστό αρκετά ικανοποιητικό, ενός αποδοτικού αλγορίθμου.

Επιπλέον, δημιουργήθηκε ένας πίνακας με τον αριθμό των συνολικών δεδομένων εκπαίδευσης και επικύρωσης, τις τιμές των υπερπαραμέτρων των επαναλήψεων και του μεγέθους της ακολουθίας που διαβάζει κάθε φορά το δίκτυο και της ακρίβειας όλων των δοκιμών που υλοποιήθηκαν για την παρούσα εργασία. Ο πίνακας είναι ο εξής:

Κώδικας	Συνολικά δεδομένα εκπαίδευσης	Συνολικά δεδομένα επικύρωσης	Μέγεθος ακολουθίας ανά φορά	Επαναλήψεις	Ακρίβεια
unidirectional_rnn	156	30	15	25000	76.67%
bidirectional_rnn	51	30	15	25000	66.67%
bidirectional_rnn	156	30	15	25000	86.67%
unidirectional_lstm_30	156	30	30	25000	83.33%
unidirectional_lstm_60	156	30	60	25000	86.67%
bidirectional_lstm_30	156	30	30	25000	90%
bidirectional_lstm_60	156	30	60	25000	93.33%
bidirectional_lstm_30	156	30	30	50000	86.67%
bidirectional_lstm_60	156	30	60	50000	90%
bidirectional_lstm_30	156	30	30	75000	76.67%
bidirectional_lstm_60	156	30	60	75000	90%

Εικόνα 37: Πίνακας με τις τιμές των παραμέτρων στις δοκιμές που υλοποιήθηκαν.

Ο πίνακας αυτός καθιστά εύκολη τη μελέτη και τη σύγκριση όσων αναλύθηκαν παραπάνω.

5.5 Αναπαράσταση και εξήγηση των κωδίκων

Αναλυτικά, η επεξήγηση των κωδίκων έχει ως εξής:

Κώδικας «bidirectional_rnn»

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 1 έως 4, γίνεται η δήλωση των βιβλιοθηκών που θα χρησιμοποιηθούν στον υπόλοιπο κώδικα. Βιβλιοθήκη καλείται μία συλλογή από έτοιμα υποπρογράμματα που χρησιμοποιείται για την ανάπτυξη λογισμικού και εξαλείφει την ανάγκη για δημιουργία κωδίκων από την αρχή. Ειδικότερα, στη γραμμή 1 γίνεται η εγκατάσταση της βιβλιοθήκης Unidecode, η οποία επιτρέπει να μετατρέψουμε τα κείμενα σε χαρακτήρες ASCII. Ο κώδικας ASCII είναι ένα κωδικοποιημένο σύνολο χαρακτήρων του λατινικού αλφάβητου και περιλαμβάνει ορισμούς για 128 χαρακτήρες. Στη συνέχεια, στη γραμμή 2, γίνεται η δήλωση του πακέτου unidecode, για τον σκοπό που περιγράψαμε παραπάνω. Στη γραμμή 3, γίνεται η δήλωση του πακέτου torch. Το πακέτο αυτό περιέχει δομές δεδομένων για πολυδιάστατους τανυστές και ορίζει μαθηματικές πράξεις πάνω σε αυτούς. Υποστηρίζει βασικές ρουτίνες για ανάκτηση του ν-οστού στοιχείου με τη χρήση δείκτη μέσα σε ένα σειριακό τύπο δεδομένων (indexing), εξαγωγή ενός υποσυνόλου στοιχείων από έναν πίνακα/τανυστή και η αποθήκευσή του ως άλλον πίνακα/τανυστή (slicing) κτλ. που είναι απαραίτητες. Στη γραμμή 4, γίνεται δήλωση του δομοστοιχείου (module) random, το οποίο υλοποιεί γεννήτριες ψευδοτυχαίων αριθμών για διάφορες κατανομές. Για ακέραιους αριθμούς, υπάρχει μία ομοιόμορφη επιλογή από ένα πεδίο τιμών.

```
1. !pip install unidecode
2. from unidecode import unidecode
3. import torch
4. import random
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 5 έως 12, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο κάποιον χαρακτήρα του μαθηματικού κειμένου και επιστρέφει την κατηγορία στην οποία ανήκει. Δηλαδή, επιστρέφει «1» αν ο χαρακτήρας είναι υποδιαστολή, «2» αν είναι τελεία και «3» αν είναι οποιοσδήποτε άλλος χαρακτήρας.

```
5. def char2cat(ch):
6.    if (ch=='·'):
7.     cat = 1;
8.    elif (ch == '.'):
9.     cat = 2;
10.   else:
11.     cat = 3;
12.   return cat;
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 13 έως 25, αναπτύσσεται συνάρτηση η οποία διαβάζει ένα TeX αρχείο με κατάλληλη κωδικοποίηση της τελείας και της υποδιαστολής. Συγκεκριμένα, στη γραμμή 14 η συνάρτηση open ανοίγει το αρχείο για να διαβαστεί, χρησιμοποιώντας κωδικοποίηση UTF-8, ένα μη-απωλεστικό σχήμα κωδικοποίησης χαρακτήρων μεταβλητού μήκους για το πρότυπο Unicode. Στις γραμμές 15 έως 23, η συνάρτηση creatematrices δημιουργεί δύο διανύσματα, ένα με τους χαρακτήρες και ένα με τις κατηγορίες στις αντίστοιχες θέσεις. Στις γραμμές 21 έως 23, παρατηρούμε ότι μετατρέπει το σύμβολο «·» σε «.» και, έπειτα, το προσαρτεί στο διάνυσμα των χαρακτήρων. Στη γραμμή 24, το αρχείο κλείνει. Στη γραμμή 25 επιστρέφονται τα διανύσματα που δημιουργήθηκαν.

```
13.   def creatematrices(filename):
14.     infile=open(filename, mode="r", encoding="utf8");
15.     characters = [];
```

```
16.     categories=[];
17.     for line in infile:
18.        for col in range(0,len(line)):
19.          ch = line[col];
20.          categories.append(char2cat(line[col]));
21.          if (ch=='·'):
22.            ch='.';
23.          characters.append(ch);
24.     infile.close();
25.     return characters,categories;
```

Στις γραμμές 26 και 27, γίνεται προσάρτηση (mount) του φακέλου στο drive με τα αρχεία εκπαίδευσης/ελέγχου.

```
26.   from google.colab import drive
27.   drive.mount('/content/drive')
```

Στη γραμμή 28, γίνεται η δημιουργία πίνακα χαρακτήρων και πίνακα κατηγοριών από τα παραδείγματα μαθηματικών κειμένων που κατασκευάσαμε.

```
28.   characters,categories = creatematrices('/content/drive/MyDri
   ve/Colab Notebooks/dimitra/examples.txt')
```

Στη γραμμή 29, ως ένα παράδειγμα προς το χρήστη, το πρόγραμμα εκτυπώνει τους πρώτους 20 χαρακτήρες του πίνακα χαρακτήρων που δημιουργήθηκε παραπάνω.

```
29.   print(characters[0:20])
```

Όμοια, στη γραμμή 30, εκτυπώνει τις πρώτες 20 αντίστοιχες κατηγορίες από τον πίνακα κατηγοριών.

```
30.   print(categories[0:20])
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 31 έως 36, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο μία ακολουθία. Στη συνέχεια, τη μετατρέπει σε τανυστή με διαστάσεις <μήκος_ακολουθίας x 128> (επιλέγουμε 128, δηλαδή όσοι είναι οι χαρακτήρες που περιλαμβάνονται στον κώδικα ASCII) ή, αλλιώς, έναν πίνακα με one-hot διανύσματα χαρακτήρων. Ένα διάνυσμα one-hot απαρτίζεται από μηδενικά σε όλες του τις θέσεις, εκτός από αυτή του εν λόγω γράμματος που παίρνει την τιμή 1. Για παράδειγμα, "b" = <0 1 0 0 0 ...>. Στη γραμμή 36, επιστρέφει αυτόν τον τανυστή.

```
31.   def charseq2tensor(seq):
32.       tensor = torch.zeros(len(seq),128)
33.       for li, letter in enumerate(seq):
34.           ch = unidecode(letter)
35.           tensor[li][ord(ch[0])-1] = 1
36.       return tensor
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 37 έως 41, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο μία ακολουθία. Στη συνέχεια, τη μετατρέπει σε τανυστή με διαστάσεις <μήκος_ακολουθίας x 3> (επιλέγουμε 3, δηλαδή όσες είναι οι κατηγορίες) ή, αλλιώς, έναν πίνακα με one-hot διανύσματα κατηγοριών. Στη γραμμή 41, επιστρέφει αυτόν τον τανυστή.

```
37.   def catseq2tensor(seq):
38.       tensor = torch.zeros(len(seq),3)
39.       for li, cat in enumerate(seq):
40.           tensor[li][cat-1] = 1
41.       return tensor
```

Στις γραμμές 42 και 43, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο έναν πίνακα (array) των μαθηματικών κειμένων με τον οποίο θα ασχοληθεί, μία θέση (pos) στον πίνακα και το μέγεθος μίας ακολουθίας (seq_size). Μετατρέπει το κομμάτι του πίνακα από τη δοθείσα θέση έως τη δοθείσα θέση συν το μέγεθος της ακολουθίας σε τανυστή, χρησιμοποιώντας τη συνάρτηση charseq2tensor που αναλύσαμε παραπάνω και επιστρέφει ως έξοδο τον τανυστή αυτόν.

```
42.   def getInputCharTensor(array, pos, seq_size):
43.     return charseq2tensor(array[pos:seq_size+pos])
```

Όμοια, στις γραμμές 44 και 45, αναπτύσσεται συνάρτηση η οποία δέχεται ως είσοδο έναν πίνακα (array) των μαθηματικών κειμένων με τον οποίο θα ασχοληθεί, μία θέση (pos) στον πίνακα και το μέγεθος μίας ακολουθίας (seq_size). Μετατρέπει το κομμάτι του πίνακα από τη δοθείσα θέση έως τη δοθείσα θέση συν το μέγεθος της ακολουθίας σε τανυστή, χρησιμοποιώντας τη συνάρτηση catseq2tensor που αναλύσαμε παραπάνω και επιστρέφει ως έξοδο τον τανυστή αυτόν. Ουσιαστικά, αυτό που κάνουν αυτή και η προηγούμενη συνάρτηση, είναι να διαβάζουν μία ακολουθία και να επιστρέφουν όλες τις υπακολουθίες με μέγεθος seq_size στη σειρά.

```
44.   def getCatTensor(array, pos, seq_size):
45.     return catseq2tensor(array[pos:seq_size+pos])
```

Στις γραμμές 46 έως 48, αναπτύσσεται συνάρτηση η οποία με δεδομένο το μέγεθος μίας ακολουθίας, που δίνεται ως είσοδος, επιστρέφει ένα τυχαίο δείγμα τανυστών χαρακτήρων και τους αντίστοιχους τανυστές κατηγοριών.

```
46.   def getRandomSample(chararray, catarray, seq_size):
47.     pos = random.randint(0, len(chararray) - seq_size)
48.     return charseq2tensor(chararray[pos:seq_size+pos]), catseq
   2tensor(catarray[pos:seq_size+pos])
```

Στη γραμμή 49, γίνεται εισαγωγή της κλάσης nn από το πακέτο torch. Γενικά, η PyTorch παρέχει το δομοστοιχείο torch.nn για να βοηθήσει στη δημιουργία και εκπαίδευση νευρωνικών δικτύων.

```
49.   from torch import nn
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 50 έως 57, χρησιμοποιείται μία λειτουργία της PyTorch για να ελεγχθεί αν η συσκευή στην οποία εκτελούμε το πρόγραμμα διαθέτει CPU ή GPU. Η CPU (central processing unit / κεντρική μονάδα επεξεργασίας) είναι ένας γενικευμένος επεξεργαστής που έχει σχεδιαστεί για να εκτελεί μία μεγάλη ποικιλία εργασιών. Η GPU (graphics processing unit / μονάδα επεξεργασίας γραφικών) είναι μία εξειδικευμένη μονάδα επεξεργασίας με αυξημένη ικανότητα μαθηματικών υπολογισμών, ιδανική για γραφικά υπολογιστών και εργασίες μηχανικής μάθησης. Επομένως, όταν το σύνολο των δεδομένων εκπαίδευσης είναι πολύ μεγάλο και το μοντέλο έχει εκατομμύρια εκπαιδευόμενες παραμέτρους, η χρήση της GPU είναι αρκετά σημαντική για να επιταχυνθεί η εκπαίδευση. Η υλοποίηση του νευρωνικού δικτύου της παρούσας εργασίας δεν απαιτεί GPU, καθώς η εκπαίδευση του δικτύου δεν είναι πολύ περίπλοκη. Συγκεκριμένα, στη γραμμή 50, η συνάρτηση torch.cuda.is_available() ελέγχει και επιστρέφει την Boolean τιμή «Αληθές» (True) αν είναι διαθέσιμη GPU, διαφορετικά επιστρέφει την Boolean τιμή «Ψευδές» (False). Στη γραμμή 53, υπό τη συνθήκη ότι GPU είναι διαθέσιμη, θέτουμε τη συσκευή μας σε GPU, ενώ αν GPU δεν είναι διαθέσιμη, στη γραμμή 56 θέτουμε τη συσκευή μας σε CPU. Η συσκευή θα χρησιμοποιηθεί στο πρόγραμμα παρακάτω. Και στις δύο περιπτώσεις εκτυπώνεται το κατάλληλο μήνυμα ώστε να ενημερωθεί ο χρήστης ποια μονάδα επεξεργασίας διαθέτει η συσκευή του (γραμμές 54 και 57).

```
50.   is_cuda = torch.cuda.is_available()
51.
```

```
52.   if is_cuda:
53.       device = torch.device("cuda")
54.       print("GPU is available")
55.   else:
56.       device = torch.device("cpu")
57.       print("GPU not available, CPU used")
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 58 έως 80, κατασκευάζεται το μοντέλο του νευρωνικού συστήματος. Για να γίνει αυτό, στη γραμμή 58, ορίζεται μία κλάση η οποία «κληρονομεί» τη βασική κλάση της PyTorch (nn.Module) για όλα τα δομοστοιχεία νευρωνικού δικτύου. Στις γραμμές 62 και 63, ορίζονται οι παράμετροι της διάστασης του κρυφού επιπέδου και του αριθμού των επιπέδων αντίστοιχα. Στις γραμμές 65 και 66, ορίζονται τα επίπεδα του μοντέλου. Για το μοντέλο αυτό, θα χρειαστεί 1 επίπεδο επαναλαμβανόμενου νευρωνικού δικτύου και, ακολούθως, ένα πλήρως συνδεδεμένο επίπεδο. Το πλήρως συνδεδεμένο επίπεδο είναι υπεύθυνο για τη μετατροπή της εξόδου του επαναλαμβανόμενου νευρωνικού δικτύου στο επιθυμητό σχήμα εξόδου. Επιπλέον, στη γραμμή 65, με την εντολή bidirectional=True, το επαναλαμβανόμενο νευρωνικό δίκτυο ορίζεται ως αμφίδρομο. Για αυτόν το λόγο, στη γραμμή 66, πολλαπλασιάζεται με 2 η διάσταση του κρυφού επιπέδου. Στις γραμμές 68 έως 76, ορίζεται η συνάρτηση forward pass με το όνομα forward() ως μία μέθοδος της κλάσης. Στη συνάρτηση αυτή, περνάμε, πρώτα (γραμμή 71), τα δεδομένα εισόδου και την κατάσταση του κρυφού επιπέδου, η οποία έχει αρχικοποιηθεί με μηδενικά (γραμμή 69) με κλήση μεθόδου που θα αναλυθεί παρακάτω, μέσα στο επίπεδο του επαναλαμβανόμενου νευρωνικού δικτύου. Στη συνέχεια, περνάμε τα δεδομένα εξόδου που εξήγαγε το επαναλαμβανόμενο νευρωνικό δίκτυο στο πλήρως συνδεδεμένο επίπεδο. Στη γραμμή 73, γίνεται αλλαγή του σχήματος των δεδομένων εξόδου έτσι ώστε να είναι κατάλληλο για το πλήρως συνδεδεμένο δίκτυο. Εδώ πολλαπλασιάζεται ξανά με 2 η διάσταση του κρυφού επιπέδου, αφού το δίκτυο είναι αμφίδρομο. Αξίζει να σημειωθεί ότι χρησιμοποιούνται τα επίπεδα που ορίστηκαν παραπάνω. Στις γραμμές 78 έως 80, ορίζεται η μέθοδος init_hidden(), την οποία καλέσαμε νωρίτερα για την αρχικοποίηση του κρυφού επιπέδου. Η μέθοδος αυτή δημιουργεί έναν τανυστή με μηδενικά στο σχήμα των κρυφών επιπέδων, ο οποίος χρησιμοποιείται ως η πρώτη κρυφή κατάσταση στο forward pass. Εδώ, πολλαπλασιάζεται με 2 ο αριθμός των επιπέδων, αφού το δίκτυο είναι αμφίδρομο.

```
58.   class Model(nn.Module):
59.       def __init__(self, input_size, output_size, hidden_dim,
   n_layers):
60.           super(Model, self).__init__()
61.
62.           self.hidden_dim = hidden_dim
63.           self.n_layers = n_layers
64.
65.           self.rnn = nn.RNN(input_size, hidden_dim, n_layers,
   bidirectional=True)
66.           self.fc = nn.Linear(2*hidden_dim, output_size)
67.
68.   def forward(self, x):
69.           hidden = self.init_hidden()
70.
71.           out, hidden = self.rnn(x, hidden)
72.
73.           out = out.contiguous().view(-1, 2*self.hidden_dim)
74.           out = self.fc(out)
```

```
75.
76.             return out, hidden
77.
78.         def init_hidden(self):
79.             hidden = torch.zeros(2*self.n_layers, self.hidden_di
    m)
80.             return hidden
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 81 έως 94, γίνεται αρχικοποίηση του μοντέλου με τις σχετικές παραμέτρους, καθώς και ορισμός των υπερπαραμέτρων. Πριν από αυτά, όμως, γίνονται κάποιες αναθέσεις τιμών. Στη γραμμή 81, η μεταβλητή characters_size παίρνει την τιμή 128, δηλαδή όσοι είναι οι χαρακτήρες που περιλαμβάνονται στον κώδικα ASCII. Στη γραμμή 82, η μεταβλητή category_size παίρνει την τιμή 3, δηλαδή όσες είναι οι κατηγορίες στις οποίες θέλουμε να ταξινομήσουμε τους χαρακτήρες των μαθηματικών κειμένων. Στη γραμμή 83, η μεταβλητή seq_size παίρνει την τιμή 15, δηλαδή η ακολουθία που δίνεται κάθε φορά στο δίκτυο αποτελείται από 15 χαρακτήρες. Η τιμή αυτή προκύπτει έπειτα από πειραματισμό του χρήστη σχετικά με το ποια τιμή δίνει τα καλύτερα αποτελέσματα, δεν είναι δηλαδή κάποια φιξαρισμένη τιμή. Στη γραμμή 84, η μεταβλητή hidden_dimension παίρνει την τιμή 12. Η τιμή αυτή, επίσης προκύπτει έπειτα από πειραματισμό του χρήστη σχετικά με το ποια τιμή δίνει τα καλύτερα αποτελέσματα. Στη γραμμή 86, γίνεται αρχικοποίηση του μοντέλου. Στη γραμμή 87, δίνεται η εντολή ώστε το μοντέλο να είναι στη συσκευή που ορίσαμε παραπάνω. Στις γραμμές 89 και 90, ορίζονται οι υπερπαράμετροι των αριθμών των επαναλήψεων τροφοδότησης παραδειγμάτων στο δίκτυο και του ρυθμού εκμάθησης, ο οποίος επηρεάζει τον ρυθμό με τον οποίο το μοντέλο κάνει ενημέρωση των βαρών στα κελιά κάθε φορά που πραγματοποιείται οπισθοδιάδοση, αντίστοιχα. Στη γραμμή 92, επιλέγεται ως συνάρτηση απώλειας η διασταυρούμενη εντροπία, διότι το τελευταίο αποτέλεσμα εξόδου είναι ουσιαστικά μία εργασία ταξινόμησης. Στη γραμμή 94, ορίζεται ως βελτιστοποιητής ο adam.

```
81.   characters_size=128
82.   category_size = 3
83.   seq_size=15
84.   hidden_dimension=12
85.
86.   model = Model(input_size=characters_size, output_size=catego
    ry_size, hidden_dim=hidden_dimension, n_layers=1)
87.   model.to(device)
88.
89.   n_epochs = 25000
90.   lr=0.001
91.
92.   criterion = nn.CrossEntropyLoss()
93.
94.   optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 101 έως 118, γίνεται η εκπαίδευση του δικτύου. Η εκπαίδευση διαρκεί για τόσα βήματα όσος ο αριθμός των επαναλήψεων που ορίστηκε παραπάνω, όπως διαπιστώνεται στη γραμμή 101. Η συγκεκριμένη εκπαίδευση δε γίνεται σε παρτίδες (unbatched learning). Στη γραμμή 102, η εντολή optimizer.zero_grad() «καθαρίζει» κάθε φορά τις υπάρχουσες κλίσεις από την προηγούμενη επανάληψη. Στη γραμμή 103, καλείται η συνάρτηση getRandomSample() που ορίστηκε παραπάνω, η οποία δίνει ένα τυχαίο δείγμα τανυστών χαρακτήρων στη μεταβλητή input_seq και τους αντίστοιχους τανυστές κατηγοριών στη μεταβλητή target_seq. Στη γραμμή 105, δίνεται η

εντολή ώστε η μεταβλητή input_seq να είναι στη συσκευή που ορίσαμε παραπάνω. Στη γραμμή 106, εισάγοντας στο μοντέλο τη μεταβλητή input_seq, γίνεται ανάθεση των τιμών των αποτελεσμάτων που εξάγει στις μεταβλητές output και hidden, αντίστοιχα. Στη γραμμή 108, γίνεται ανάθεση της τιμής που εξάγεται με τη βοήθεια της διασταυρούμενης εντροπίας στη μεταβλητή loss. Στη γραμμή 109, με την εντολή loss.backward() πραγματοποιείται οπισθοδιάδοση και υπολογισμός των κλίσεων. Στη γραμμή 110, με την εντολή optimizer.step() γίνεται η κατάλληλη ενημέρωση των βαρών. Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 115 έως 117, δίνονται οι εντολές ώστε να εκτυπώνεται στην οθόνη ανά 500 επαναλήψεις ο τρέχων αριθμός επανάληψης και η αντίστοιχη απώλεια. Ο χρήστης καταλαβαίνει ότι η εκπαίδευση πραγματοποιείται επιτυχώς, όταν η απώλεια αυτή μειώνεται καθώς εκτελούνται όλο και περισσότερες επαναλήψεις. Στο κομμάτι αυτό, επίσης, δίνονται οι κατάλληλες εντολές, ώστε να εμφανίζεται στην οθόνη η γραφική παράσταση για το σφάλμα του δικτύου. Στη γραμμή 95, γίνεται εισαγωγή στο πρόγραμμα της συλλογής συναρτήσεων pyplot του δημοφιλούς πακέτου απεικονίσεων Matplotlib. Γενικά, κάθε συνάρτηση pyplot κάνει κάποιες αλλαγές σε μία εικόνα, για παράδειγμα δημιουργεί μία εικόνα, δημιουργεί μία περιοχή σχεδίασης στην εικόνα αυτή, σχεδιάζει μερικές γραμμές σε μία περιοχή σχεδίασης, τοποθετεί ετικέτες στη γραφική παράσταση κτλ. Στις γραμμές 97 και 98, δημιουργούνται τα διανύσματα για την απώλεια και τις επαναλήψεις αντίστοιχα. Στη γραμμή 99, αρχικοποιείται ο αριθμός των επαναλήψεων n. Στις γραμμές 112 και 113, αποθηκεύονται οι τρέχουσες πληροφορίες για τις επαναλήψεις και την απώλεια για κάθε επανάληψη. Στη γραμμή 113, υπολογίζεται η μέση απώλεια. Στη γραμμή 118, αυξάνεται ο αριθμός n για κάθε επανάληψη. Στις γραμμές 120 έως και 124, δίνονται οι εντολές οι οποίες θα εμφανίσουν στην οθόνη τη γραφική παράσταση. Δίνεται ο τίτλος της γραφικής παράστασης (γραμμή 120), οι τιμές της και η ονομασία της γραμμής της (γραμμή 121), η ονομασία του άξονα x (γραμμή 122), η ονομασία του άξονα y (γραμμή 123) και η εντολή εμφάνισης της γραφικής παράστασης (γραμμή 124).

```
95.   import matplotlib.pyplot as plt
96.
97.   losses = []
98.   iters = []
99.   n=0
100.
101.  for epoch in range(1, n_epochs + 1):
102.      optimizer.zero_grad()
103.      input_seq, target_seq = getRandomSample(characters,categ
   ories,seq_size)
104.
105.      input_seq.to(device)
106.      output, hidden = model(input_seq)
107.
108.      loss = criterion(output, target_seq)
109.      loss.backward()
110.      optimizer.step()
111.
112.      iters.append(n)
113.      losses.append(float(loss)/500)
114.
115.      if epoch%500 == 0:
116.          print('Epoch: {}/{}.............'.format(epoch, n_ep
   ochs), end=' ')
```

```
117.          print("Loss: {:.8f}".format(loss.item()))
118.      n += 1
119.
120.  plt.title("Training Curve (learning rate={})".format(lr))
121.  plt.plot(iters, losses, label="Training Loss")
122.  plt.xlabel("Iterations")
123.  plt.ylabel("Loss")
124.  plt.show()
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 125 έως 166, πραγματοποιείται ο έλεγχος του μοντέλου, όπου ο χρήστης μπορεί να αξιολογήσει τα αποτελέσματα που θα εξάγει το πρόγραμμα. Στη γραμμή 125, ορίζεται η συνάρτηση predict(), η οποία δέχεται ως είσοδο μία πρόταση. Μέσα στη συνάρτηση αυτή, δημιουργείται το διάνυσμα list_of_chars, το οποίο «γεμίζει» με τους χαρακτήρες της προαναφερθείσας πρότασης (γραμμή 129). Στη γραμμή 130, το διάνυσμα αυτό μετατρέπεται σε τανυστή, με κλήση της συνάρτησης charseq2tensor() που περιγράψαμε παραπάνω, και γίνεται ανάθεση αυτού στη μεταβλητή input. Στη γραμμή 132, εισάγοντας τη μεταβλητή αυτή στο μοντέλο, παίρνουμε την έξοδο του μοντέλου και αναθέτουμε την τιμή της στη μεταβλητή out. Στις γραμμές 134 έως 138, υλοποιείται επαναλαμβανόμενη εκτέλεση εντολών για κάθε χαρακτήρα της πρότασης. Στη γραμμή 135, μετατρέπεται σε λίστα το στοιχείο της εξόδου του μοντέλου στη θέση pos, στη θέση δηλαδή που αντιστοιχεί στον εκάστοτε χαρακτήρα της επανάληψης, και γίνεται ανάθεση αυτής στη μεταβλητή lst. Στη γραμμή 136, γίνεται ανάθεση του μεγαλύτερου στοιχείου της λίστας αυτής στη μεταβλητή mx. Στη γραμμή 137, προστίθεται μία μονάδα στο δείκτη της μεταβλητής mx στη λίστα lst και γίνεται ανάθεση αυτής της τιμής στη μεταβλητή cat. Στη γραμμή 138, δίνεται η εντολή εκτύπωσης στην οθόνη του εκάστοτε χαρακτήρα, ακολούθως του κενού και, τέλος, με κλήση της συνάρτησης code2cat(), της κατηγορίας στην οποία ανήκει ο χαρακτήρας. Στις γραμμές 140 έως 146, ορίζεται η συνάρτηση code2cat(), η οποία δέχεται ως είσοδο κάποια μεταβλητή code. Στην περίπτωση που η μεταβλητή αυτή είναι ο αριθμός 1, τότε η συνάρτηση επιστρέφει τη λέξη «υποδιαστολή». Από την άλλη, αν η μεταβλητή είναι ο αριθμός 2, τότε η συνάρτηση επιστρέφει τη λέξη «τελεία». Σε κάθε άλλη περίπτωση, επιστρέφει το κενό.

```
125.  def predict(sentence):
126.      list_of_chars = []
127.
128.      for i in sentence:
129.        list_of_chars.append(i)
130.      input = charseq2tensor(list_of_chars)
131.
132.      out, _= model(input)
133.
134.      for pos, ch in enumerate(sentence):
135.        lst = out[pos].tolist()
136.        mx = max(lst)
137.        cat = lst.index(mx) + 1
138.        print(ch + " " + code2cat(cat))
139.
140.  def code2cat(code):
141.    cat =""
142.    if code == 1:
143.      cat = "υποδιαστολή"
144.    elif code == 2:
145.      cat = "τελεία"
```

```
146.     return cat
```

Στο τμήμα του κώδικα που απαρτίζεται από τις γραμμές 147 έως 176, γίνονται τριάντα δοκιμές από το χρήστη, με προτάσεις που περιέχουν, μεταξύ άλλων, τελείες και υποδιαστολές, ώστε να ελέγξει την αποδοτικότητα του μοντέλου.

```
147.  predict("x=1.2.")
148.  predict("Καλημέρα.")
149.  predict("Εχουμε 1.5 χρόνο να τα πούμε.")
150.  predict("Είναι μόλις $1.5$.")
151.  predict("Θα σε δω σε 3.30 ώρες.")
152.  predict("y+z=4.67.")
153.  predict("Δώσε αποτέλεσμα.")
154.  predict("Το άθροισμα ισούται με $8.9$.")
155.  predict("Αφαίρεσε.")
156.  predict("$o=83.2.$")
157.  predict("Η δειάρκεια είναι 2.3 ώρες.")
158.  predict("$\int_{8}^{\infty}g(x)\,dx.$")
159.  predict("q=2.7.")
160.  predict("Ισούται με $2.6$.")
161.  predict("Αθροίζω.")
162.  predict("Πολλαπλασίασε με 3.2 εδώ.")
163.  predict("$$\int_{6.6}^{8}h(x)\,dx.$$")
164.  predict("Το $5.2$ μικρότερο του $4$.")
165.  predict("w=65.2")
166.  predict("e-r=7.3.")
167.  predict("Το $7$ μικρότερο του $8.9$.")
168.  predict("Το $8.2$ μεγαλύτερο του $2.5$.")
169.  predict("α-β=32.54.")
170.  predict("$f=3.2$")
171.  predict("lysi=1.4.")
172.  predict("Το $1$ μικρότερο του $2$.")
173.  predict("Θα δούμε.")
174.  predict("Εχουν μείνει μόνο 2.4 ευρώ.")
175.  predict("Το 2.1 μικρότερο του 6.")
176.  predict("$$h=[h].x_1.$$")
```

Με βάση τον κώδικα «bidirectional_rnn» και κάνοντας ορισμένες τροποποιήσεις, δημιουργήθηκε και ο κώδικας ενός αμφίδρομου δικτύου μακράς βραχύχρονης μνήμης. Οι αλλαγές έγιναν μέσα στην κλάση του μοντέλου του συστήματος. Συγκεκριμένα, τα rnn άλλαξαν σε lstm. Επίσης, αρχικοποιήθηκε ο τανυστής για την κατάσταση κελιού στο ίδιο σχήμα με την αρχικοποίηση της κατάστασης του κρυφού επιπέδου. Για το λόγο αυτό, στη γραμμή 71, εκτός από τα δεδομένα εισόδου, περάσαμε την κατάσταση του κρυφού επιπέδου και την κατάσταση του κελιού ως μία λίστα (tuple) δύο στοιχείων. Όταν τρέξαμε τον εν λόγω κώδικα, παρατηρήσαμε ότι πάθαινε υπερπροσαρμογή. Απόρροια αυτού ήταν να χρησιμοποιήσουμε τεχνικές εξομάλυνσης ώστε να λάβουμε πιο ορθά αποτελέσματα. Έτσι, στην κλάση του μοντέλου προσθέσαμε την εντολή nn.Dropout(), ώστε να προσδιορίσουμε το επιπλέον επίπεδο της dropout. Η εντολή αυτή απενεργοποιεί τυχαία κάποια από τα στοιχεία του τανυστή εισόδου κατά την εκπαίδευση. Ακόμη, για να εφαρμόσουμε L2 εξομάλυνση, χρησιμοποιήσαμε την παράμετρο weight_decay μέσα στη συνάρτηση optim.Adam(). Ειδικότερα, χρησιμοποιήσαμε το βελτιστοποιητή που ήδη είχε ο κώδικας «bidirectional_rnn»,

όπου η παράμετρος weight_decay έχει την προεπιλεγμένη τιμή 0, και προσθέσαμε τον δεύτερο βελτιστοποιητή όπου θέσαμε weight_decay =1e-4. Τέλος, αξίζει να σημειωθεί ότι αυξήσαμε τον αριθμό του seq_size από 15 σε 60, αφού παρατηρήσαμε ότι έτσι το σφάλμα του δικτύου εμφάνιζε αισθητή μείωση. Με τις αλλαγές αυτές, το πρόγραμμα ήταν σε μεγάλο βαθμό σε θέση να ξεχωρίσει την τελεία από την υποδιαστολή σε μαθηματικά κείμενα, καθώς έδωσε τα σωστά αποτελέσματα στις δοκιμές με ακρίβεια 93.33%. Παρακάτω γίνεται παράθεση του κώδικα αυτού.

Κώδικας «bidirectional_lstm»

```
1. !pip install unidecode
2. from unidecode import unidecode
3. import torch
4. import random
5. import matplotlib.pyplot as plt
6. def char2cat(ch):
7.   if (ch=='·'):
8.      cat = 1;
9.    elif (ch == '.'):
10.     cat = 2;
11.   else:
12.     cat = 3;
13.   return cat;
14.   def creatematrices(filename):
15.     infile=open(filename, mode="r", encoding="utf8");
16.     characters = [];
17.     categories=[];
18.     for line in infile:
19.        for col in range(0,len(line)):
20.          ch = line[col];
21.          categories.append(char2cat(line[col]));
22.          if (ch=='·'):
23.            ch='.';
24.          characters.append(ch);
25.     infile.close();
26.     return characters,categories;
27.   from google.colab import drive
28.   drive.mount("/content/drive", force_remount=True)
29.   characters,categories = creatematrices('/content/drive/MyDri
   ve/Colab Notebooks/diplomatiki/examplesnewnew.txt')
30.   print(characters[0:20])
31.   print(categories[0:20])
32.   def charseq2tensor(seq):
33.       tensor = torch.zeros(len(seq),128)
34.       for li, letter in enumerate(seq):
35.           ch = unidecode(letter)
36.           tensor[li][ord(ch[0])-1] = 1
37.       return tensor
38.
39.   def catseq2tensor(seq):
```

```
        tensor = torch.zeros(len(seq),3)
        for li, cat in enumerate(seq):
            tensor[li][cat-1] = 1
        return tensor

    def getInputCharTensor(array, pos, seq_size):
      return charseq2tensor(array[pos:seq_size+pos])

    def getCatTensor(array, pos, seq_size):
      return catseq2tensor(array[pos:seq_size+pos])

    def getRandomSample(chararray, catarray, seq_size):
      pos = random.randint(0, len(chararray) - seq_size)
      return charseq2tensor(chararray[pos:seq_size+pos]), catseq
  2tensor(catarray[pos:seq_size+pos])

    from torch import nn

    is_cuda = torch.cuda.is_available()
    if is_cuda:
        device = torch.device("cuda")
        print("GPU is available")
    else:
        device = torch.device("cpu")
        print("GPU not available, CPU used")

    class Model(nn.Module):
        def __init__(self, input_size, output_size, hidden_dim,
  n_layers):
            super(Model, self).__init__()

            self.hidden_dim = hidden_dim
            self.n_layers = n_layers

            nn.Dropout()

            self.lstm = nn.LSTM(input_size, hidden_dim, n_layers
  , bidirectional=True)

            self.fc = nn.Linear(2*hidden_dim, output_size)

        def forward(self, x):
            hidden = self.init_hidden()
            c0 = self.init_cell()
            out, hidden = self.lstm(x, (hidden, c0))
            out = out.contiguous().view(-1, 2*self.hidden_dim)
            out = self.fc(out)
            return out, hidden
```

```
85.
86.       def init_cell(self):
87.           c0 = torch.zeros(2*self.n_layers, self.hidden_dim)
88.           return c0
89.
90.       def init_hidden(self):
91.           hidden = torch.zeros(2*self.n_layers, self.hidden_di
   m)
92.           return hidden
93.
94.   characters_size=128
95.   category_size = 3
96.   seq_size=60
97.   hidden_dimension=12
98.
99.   model = Model(input_size=characters_size, output_size=catego
   ry_size, hidden_dim=hidden_dimension,
100.                n_layers=1)
101.  model.to(device)
102.
103.  n_epochs = 25000
104.  lr=0.001
105.
106.  criterion = nn.CrossEntropyLoss()
107.
108.  weight_decay =1e-4
109.
110.  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
111.  optimizer2 = torch.optim.Adam(model.parameters(), lr=lr, wei
   ght_decay= weight_decay)
112.
113.  losses = []
114.  iters = []
115.  n=0
116.
117.  for epoch in range(1, n_epochs + 1):
118.      optimizer.zero_grad()
119.      input_seq, target_seq = getRandomSample(characters,categ
   ories,seq_size)
120.
121.      input_seq.to(device)
122.      output, hidden = model(input_seq)
123.
124.      loss = criterion(output, target_seq)
125.      loss.backward()
126.      optimizer.step()
127.
128.      iters.append(n)
```

```
129.      losses.append(float(loss)/500)
130.
131.      if epoch%500 == 0:
132.          print('Epoch: {}/{}.............'.format(epoch, n_ep
   ochs), end=' ')
133.          print("Loss: {:.8f}".format(loss.item()))
134.      n += 1
135.
136.  plt.title("Training Curve (learning rate={})".format(lr))
137.  plt.plot(iters, losses, label="Training Loss")
138.  plt.xlabel("Iterations")
139.  plt.ylabel("Loss")
140.  plt.show()
141.
142.  def predict(sentence):
143.      list_of_chars = []
144.
145.      for i in sentence:
146.        list_of_chars.append(i)
147.      input = charseq2tensor(list_of_chars)
148.
149.      out, _= model(input)
150.
151.      for pos, ch in enumerate(sentence):
152.        lst = out[pos].tolist()
153.        mx = max(lst)
154.        cat = lst.index(mx) + 1
155.        print(ch + " " + code2cat(cat))
156.
157.  def code2cat(code):
158.    cat =""
159.    if code == 1:
160.      cat = "υποδιαστολή"
161.    elif code == 2:
162.      cat = "τελεία"
163.    return cat
164.
165.  predict("x=1.2.")
166.  predict("Καλημέρα.")
167.  predict("Εχουμε 1.5 χρόνο να τα πούμε.")
168.  predict("Είναι μόλις $1.5$.")
169.  predict("Θα σε δω σε 3.30 ώρες.")
170.  predict("y+z=4.67.")
171.  predict("Δώσε αποτέλεσμα.")
172.  predict("Το άθροισμα ισούται με $8.9$.")
173.  predict("Αφαίρεσε.")
174.  predict("$o=83.2.$")
175.  predict("Η δειάρκεια είναι 2.3 ώρες.")
```

```
predict("$\int_{8}^{\infty}g(x)\,dx.$")
predict("q=2.7.")
predict("Ισούται με $2.6$.")
predict("Αθροίζω.")
predict("Πολλαπλασίασε με 3.2 εδώ.")
predict("$$\int_{6.6}^{8}h(x)\,dx.$$")
predict("Το $5.2$ μικρότερο του $4$.")
predict("w=65.2")
predict("e-r=7.3.")
predict("Το $7$ μικρότερο του $8.9$.")
predict("Το $8.2$ μεγαλύτερο του $2.5$.")
predict("α-β=32.54.")
predict("$f=3.2$")
predict("lysi=1.4.")
predict("Το $1$ μικρότερο του $2$."
predict("Θα δούμε.")
predict("Εχουν μείνει μόνο 2.4 ευρώ.")
predict("Το 2.1 μικρότερο του 6.")
predict("$$h=[h].x_1.$$")
```

5.6 Μελλοντική εργασία

Στην ενότητα αυτή προτείνονται ιδέες με τις οποίες θα μπορούσε να συνεχιστεί η υλοποίηση της παρούσας διπλωματικής εργασίας. Αρχικά, πολύ χρήσιμη θα ήταν η προσθήκη περισσότερων παραδειγμάτων στο σύνολο δεδομένων εκπαίδευσης. Εισάγοντας τα δεδομένα αυτά στο δίκτυο μακράς βραχύχρονης μνήμης που υλοποιήσαμε, θα ελεγχθεί αν εξάγει αποτελέσματα με τόσο υψηλή ακρίβεια όση και με τα τρέχοντα δεδομένα. Όμοια, το ίδιο προτείνεται και για τα δεδομένα επικύρωσης. Κατά συνέπεια, θα είναι δυνατή και η υλοποίηση ενός βαθιού αμφίδρομου νευρωνικού δικτύου, το οποίο, σύμφωνα με το θεωρητικό μέρος της εργασίας, δίνει υψηλότερη μαθησιακή ικανότητα, αλλά χρειάζεται περισσότερα δεδομένα εκπαίδευσης. Επιπλέον, προτείνεται η αλλαγή του μεγέθους παρτίδας από το τωρινό, το οποίο είναι η μονάδα. Ωστόσο, καλή θα ήταν η αποφυγή πολύ μεγάλων μεγεθών, διότι είναι πιθανό να επηρεαστεί αρνητικά η ακρίβεια του δικτύου κατά την εκπαίδευση, αφού μειώνουν την στοχαστικότητα της κατάβασης κλίσης. Τέλος, προτείνεται η αλλαγή της παραμέτρου της μείωσης του βάρους (weight_decay), ώστε να διαπιστωθεί αν σε κάποιες τιμές το δίκτυο παθαίνει υπερπροσαρμογή ή αντιμετωπίζει το πρόβλημα της έκρηξης των κλίσεων.

ΠΑΡΑΡΤΗΜΑ

Παρατίθενται ο κώδικας για το μονόδρομο επαναλαμβανόμενο νευρωνικό δίκτυο και ο κώδικας για το μονόδρομο δίκτυο μακράς βραχύχρονης μνήμης, οι οποίοι αναλύθηκαν σε προηγούμενες ενότητες.

Κώδικας «unidirectional_rnn»

```
1. !pip install unidecode
2. from unidecode import unidecode
3. import torch
4. import random
5. import matplotlib.pyplot as plt
6.
7. def char2cat(ch):
8.    if (ch=='·'):
9.     cat = 1;
10.   elif (ch == '.'):
11.     cat = 2;
12.   else:
13.     cat = 3;
14.   return cat;
15.
16.   def creatematrices(filename):
17.     infile=open(filename, mode="r", encoding="utf8");
18.     characters = [];
19.     categories=[];
20.     for line in infile:
21.        for col in range(0,len(line)):
22.          ch = line[col];
23.          categories.append(char2cat(line[col]));
24.          if (ch=='·'):
25.            ch='.';
26.          characters.append(ch);
27.     infile.close();
28.     return characters,categories;
29.   from google.colab import drive
30.   drive.mount('/content/drive')
31.   characters,categories = creatematrices('/content/drive/MyDri
   ve/Colab Notebooks/diplomatiki/examples3.txt')
32.
33.   print(characters[0:20])
34.   print(categories[0:20])
35.
36.   def charseq2tensor(seq):
37.       tensor = torch.zeros(len(seq),128)
38.       for li, letter in enumerate(seq):
39.           ch = unidecode(letter)
40.           tensor[li][ord(ch[0])-1] = 1
41.       return tensor
```

```
42.
43.   def catseq2tensor(seq):
44.       tensor = torch.zeros(len(seq),3)
45.       for li, cat in enumerate(seq):
46.           tensor[li][cat-1] = 1
47.       return tensor
48.
49.   def getInputCharTensor(array, pos, seq_size):
50.     return charseq2tensor(array[pos:seq_size+pos])
51.
52.   def getCatTensor(array, pos, seq_size):
53.     return catseq2tensor(array[pos:seq_size+pos])
54.
55.   def getRandomSample(chararray, catarray, seq_size):
56.     pos = random.randint(0, len(chararray) - seq_size)
57.     return charseq2tensor(chararray[pos:seq_size+pos]), catseq
   2tensor(catarray[pos:seq_size+pos])
58.
59.   from torch import nn
60.
61.   is_cuda = torch.cuda.is_available()
62.
63.   if is_cuda:
64.       device = torch.device("cuda")
65.       print("GPU is available")
66.   else:
67.       device = torch.device("cpu")
68.       print("GPU not available, CPU used")
69.
70.   class Model(nn.Module):
71.       def __init__(self, input_size, output_size, hidden_dim,
   n_layers):
72.           super(Model, self).__init__()
73.
74.           self.hidden_dim = hidden_dim
75.           self.n_layers = n_layers
76.
77.           self.rnn = nn.RNN(input_size, hidden_dim, n_layers)
78.           self.fc = nn.Linear(hidden_dim, output_size)
79.
80.   def forward(self, x):
81.           hidden = self.init_hidden()
82.
83.           out, hidden = self.rnn(x, hidden)
84.
85.           out = out.contiguous().view(-1, self.hidden_dim)
86.           out = self.fc(out)
87.
```

```
88.           return out, hidden
89.
90.    def init_hidden(self):
91.           hidden = torch.zeros(self.n_layers, self.hidden_dim)
92.           return hidden
93.
94.   characters_size=128
95.   category_size = 3
96.   seq_size=15
97.   hidden_dimension=12
98.
99.   model = Model(input_size=characters_size, output_size=catego
   ry_size, hidden_dim=hidden_dimension, n_layers=1)
100.  model.to(device)
101.
102.  n_epochs = 25000
103.  lr=0.001
104.
105.  criterion = nn.CrossEntropyLoss()
106.
107.  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
108.
109.  losses = []
110.  iters = []
111.  n=0
112.
113.  for epoch in range(1, n_epochs + 1):
114.      optimizer.zero_grad()
115.      input_seq, target_seq = getRandomSample(characters,categ
   ories,seq_size)
116.
117.      input_seq.to(device)
118.      output, hidden = model(input_seq)
119.
120.      loss = criterion(output, target_seq)
121.      loss.backward()
122.      optimizer.step()
123.
124.      iters.append(n)
125.      losses.append(float(loss)/500)
126.
127.      if epoch%500 == 0:
128.          print('Epoch: {}/{}.............'.format(epoch, n_ep
   ochs), end=' ')
129.          print("Loss: {:.8f}".format(loss.item()))
130.      n += 1
131.
132.  plt.title("Training Curve (learning rate={})".format(lr))
```

```
133.  plt.plot(iters, losses, label="Training Loss")
134.  plt.xlabel("Iterations")
135.  plt.ylabel("Loss")
136.  plt.show()
137.
138.  def predict(sentence):
139.      list_of_chars = []
140.
141.      for i in sentence:
142.        list_of_chars.append(i)
143.      input = charseq2tensor(list_of_chars)
144.
145.      out, _= model(input)
146.
147.      for pos, ch in enumerate(sentence):
148.        lst = out[pos].tolist()
149.        mx = max(lst)
150.        cat = lst.index(mx) + 1
151.        print(ch + " " + code2cat(cat))
152.
153.  def code2cat(code):
154.    cat =""
155.    if code == 1:
156.      cat = "υποδιαστολή"
157.    elif code == 2:
158.      cat = "τελεία"
159.    return cat
160.
161.  predict("x=1.2.")
162.  predict("Καλημέρα.")
163.  predict("Εχουμε 1.5 χρόνο να τα πούμε.")
164.  predict("Είναι μόλις $1.5$.")
165.  predict("Θα σε δω σε 3.30 ώρες.")
166.  predict("y+z=4.67.")
167.  predict("Δώσε αποτέλεσμα.")
168.  predict("Το άθροισμα ισούται με $8.9$.")
169.  predict("Αφαίρεσε.")
170.  predict("$o=83.2.$")
171.  predict("Η διάρκεια είναι 2.3 ώρες.")
172.  predict("$\int_{8}^{\infty}g(x)\,dx.$")
173.  predict("q=2.7.")
174.  predict("Ισούται με $2.6$.")
175.  predict("Αθροίζω.")
176.  predict("Πολλαπλασίασε με 3.2 εδώ.")
177.  predict("$$\int_{6.6}^{8}h(x)\,dx.$$")
178.  predict("Το $5.2$ μικρότερο του $4$.")
179.  predict("w=65.2")
180.  predict("e-r=7.3.")
```

```
predict("Το $7$ μικρότερο του $8.9$.")
predict("Το $8.2$ μεγαλύτερο του $2.5$.")
predict("α-β=32.54.")
predict("$f=3.2$")
predict("lysi=1.4.")
predict("Το $1$ μικρότερο του $2$."
predict("Θα δούμε.")
predict("Εχουν μείνει μόνο 2.4 ευρώ.")
predict("Το 2.1 μικρότερο του 6.")
predict("$$h=[h].x_1.$$")
```

Κώδικας «unidirectional_lstm_60»

```
!pip install unidecode
from unidecode import unidecode
import torch
import random
import matplotlib.pyplot as plt

def char2cat(ch):
    if (ch=='·'):
     cat = 1;
    elif (ch == '.'):
      cat = 2;
    else:
      cat = 3;
    return cat;

    def creatematrices(filename):
      infile=open(filename, mode="r", encoding="utf8");
      characters = [];
      categories=[];
      for line in infile:
         for col in range(0,len(line)):
           ch = line[col];
           categories.append(char2cat(line[col]));
           if (ch=='·'):
             ch='.';
           characters.append(ch);
      infile.close();
      return characters,categories;

    from google.colab import drive
    drive.mount('/content/drive')
    characters,categories = creatematrices('/content/drive/MyDrive/Colab Notebooks/diplomatiki/examples3.txt')

    print(characters[0:20])
    print(categories[0:20])
```

```
36.
37.   def charseq2tensor(seq):
38.       tensor = torch.zeros(len(seq),128)
39.       for li, letter in enumerate(seq):
40.           ch = unidecode(letter)
41.           tensor[li][ord(ch[0])-1] = 1
42.       return tensor
43.
44.   def catseq2tensor(seq):
45.       tensor = torch.zeros(len(seq),3)
46.       for li, cat in enumerate(seq):
47.           tensor[li][cat-1] = 1
48.       return tensor
49.
50.   def getInputCharTensor(array, pos, seq_size):
51.     return charseq2tensor(array[pos:seq_size+pos])
52.
53.   def getCatTensor(array, pos, seq_size):
54.     return catseq2tensor(array[pos:seq_size+pos])
55.
56.   def getRandomSample(chararray, catarray, seq_size):
57.     pos = random.randint(0, len(chararray) - seq_size)
58.     return charseq2tensor(chararray[pos:seq_size+pos]), catseq
   2tensor(catarray[pos:seq_size+pos])
59.
60.   from torch import nn
61.
62.   is_cuda = torch.cuda.is_available()
63.
64.   if is_cuda:
65.       device = torch.device("cuda")
66.       print("GPU is available")
67.   else:
68.       device = torch.device("cpu")
69.       print("GPU not available, CPU used")
70.
71.   class Model(nn.Module):
72.       def __init__(self, input_size, output_size, hidden_dim,
   n_layers):
73.           super(Model, self).__init__()
74.
75.           self.hidden_dim = hidden_dim
76.           self.n_layers = n_layers
77.
78.           nn.Dropout()
79.
80.           self.lstm = nn.LSTM(input_size, hidden_dim, n_layers
   )
```

```
81.           self.fc = nn.Linear(hidden_dim, output_size)
82.
83.   def forward(self, x):
84.           hidden = self.init_hidden()
85.           c0 = self.init_cell()
86.
87.           out, hidden = self.lstm(x, (hidden, c0))
88.
89.           out = out.contiguous().view(-1, self.hidden_dim)
90.           out = self.fc(out)
91.
92.           return out, hidden
93.
94.       def init_cell(self):
95.           c0 = torch.zeros(self.n_layers, self.hidden_dim)
96.           return c0
97.
98.    def init_hidden(self):
99.           hidden = torch.zeros(self.n_layers, self.hidden_dim)
100.          return hidden
101.
102.  characters_size=128
103.  category_size = 3
104.  seq_size=60
105.  hidden_dimension=12
106.
107.  model = Model(input_size=characters_size, output_size=catego
   ry_size, hidden_dim=hidden_dimension, n_layers=1)
108.  model.to(device)
109.
110.  n_epochs = 25000
111.  lr=0.001
112.
113.  criterion = nn.CrossEntropyLoss()
114.
115.  optimizer = torch.optim.Adam(model.parameters(), lr=lr)
116.
117.  losses = []
118.  iters = []
119.  n=0
120.
121.  for epoch in range(1, n_epochs + 1):
122.      optimizer.zero_grad()
123.      input_seq, target_seq = getRandomSample(characters,categ
   ories,seq_size)
124.
125.      input_seq.to(device)
126.      output, hidden = model(input_seq)
```

```
127.
128.       loss = criterion(output, target_seq)
129.       loss.backward()
130.       optimizer.step()
131.
132.       iters.append(n)
133.       losses.append(float(loss)/500)
134.
135.       if epoch%500 == 0:
136.           print('Epoch: {}/{}.............'.format(epoch, n_ep
    ochs), end=' ')
137.           print("Loss: {:.8f}".format(loss.item()))
138.       n += 1
139.
140.   plt.title("Training Curve (learning rate={})".format(lr))
141.   plt.plot(iters, losses, label="Training Loss")
142.   plt.xlabel("Iterations")
143.   plt.ylabel("Loss")
144.   plt.show()
145.
146.   def predict(sentence):
147.       list_of_chars = []
148.
149.       for i in sentence:
150.         list_of_chars.append(i)
151.       input = charseq2tensor(list_of_chars)
152.
153.       out, _= model(input)
154.
155.       for pos, ch in enumerate(sentence):
156.         lst = out[pos].tolist()
157.         mx = max(lst)
158.         cat = lst.index(mx) + 1
159.         print(ch + " " + code2cat(cat))
160.
161.   def code2cat(code):
162.     cat =""
163.     if code == 1:
164.       cat = "υποδιαστολή"
165.     elif code == 2:
166.       cat = "τελεία"
167.     return cat
168.
169.   predict("x=1.2.")
170.   predict("Καλημέρα.")
171.   predict("Εχουμε 1.5 χρόνο να τα πούμε.")
172.   predict("Είναι μόλις $1.5$.")
173.   predict("Θα σε δω σε 3.30 ώρες.")
```

```
174.  predict("y+z=4.67.")
175.  predict("Δώσε αποτέλεσμα.")
176.  predict("Το άθροισμα ισούται με $8.9$.")
177.  predict("Αφαίρεσε.")
178.  predict("$o=83.2.$")
179.  predict("Η δειάρκεια είναι 2.3 ώρες.")
180.  predict("$\int_{8}^{\infty}g(x)\,dx.$")
181.  predict("q=2.7.")
182.  predict("Ισούται με $2.6$.")
183.  predict("Αθροίζω.")
184.  predict("Πολλαπλασίασε με 3.2 εδώ.")
185.  predict("$$\int_{6.6}^{8}h(x)\,dx.$$")
186.  predict("Το $5.2$ μικρότερο του $4$.")
187.  predict("w=65.2")
188.  predict("e-r=7.3.")
189.  predict("Το $7$ μικρότερο του $8.9$.")
190.  predict("Το $8.2$ μεγαλύτερο του $2.5$.")
191.  predict("α-β=32.54.")
192.  predict("$f=3.2$")
193.  predict("lysi=1.4.")
194.  predict("Το $1$ μικρότερο του $2$."
195.  predict("Θα δούμε.")
196.  predict("Εχουν μείνει μόνο 2.4 ευρώ.")
197.  predict("Το 2.1 μικρότερο του 6.")
198.  predict("$$h=[h].x_1.$$")
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Barbara Hammer, “Learning with Recurrent Neural Networks”, 1999. Διαθέσιμο στο: https://www.researchgate.net/publication/2273271_Learning_with_Recurrent_Neural_Networks
2. Wikipedia, Machine learning, έκδοση της 7ης Μαρτίου 2022. Διαθέσιμο στο: Machine learning - Wikipedia
3. Wikipedia, Artificial neural network, έκδοση της 5ης Μαρτίου 2022. Διαθέσιμο στο: Artificial neural network - Wikipedia
4. Wikipedia, Frank Rosenblatt, έκδοση της 5ης Μαρτίου 2022. Διαθέσιμο στο: https://en.wikipedia.org/wiki/Frank_Rosenblatt
5. Siddharth Pandey, “An introduction to Machine Learning”, 2021. Διαθέσιμο στο: https://www.geeksforgeeks.org/introduction-machine-learning/
6. K.-L. Du and M.N.s. Swamy, “Neural Networks and Statistical Learning”, 2014. Διαθέσιμο στο: https://www.researchgate.net/publication/278654277_Neural_Networks_and_Statistical_Learning
7. Wikipedia, Learning rate, έκδοση της 30ης Ιανουαρίου 2022. Διαθέσιμο στο: https://en.wikipedia.org/wiki/Learning_rate
8. Jason Brownlee, “A Gentle Introduction to Cross-Entropy for Machine Learning”, 2019. Διαθέσιμο στο: https://machinelearningmastery.com/cross-entropy-for-machine-learning/
9. Strahinja Zivkovic, “Popular techniques to prevent the Overfitting in a Neural Networks”, 2021. Διαθέσιμο στο: https://datahacker.rs/018-pytorch-popular-techniques-to-prevent-the-overfitting-in-a-neural-networks/
10. Ayush Gupta, “A Comprehensive Guide on Deep Learning Optimizers”, 2021. Διαθέσιμο στο: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/
11. Michael Nielsen, “Neural Networks and Deep Learning”, Dec 2019. Διαθέσιμο στο: http://neuralnetworksanddeeplearning.com/
12. Ian Goodfellow, Yoshua Bengio and Aaron Courville, “Deep Learning, An MIT Press book”, 2016, Part I Chapter 5, Part II Chapter 8, Chapter 10, Chapter 12. Διαθέσιμο στο: https://www.deeplearningbook.org/
13. Pablo Caceres, “The Recurrent Neural Network - Theory and Implementation of the Elman Network and LSTM”, 2020. Διαθέσιμο στο: https://pabloinsente.github.io/the-recurrent-net
14. Christopher Olah, “Understanding LSTM Networks”, 2015. Διαθέσιμο στο: https://colah.github.io/posts/2015-08-Understanding-LSTMs/
15. Felix Gers, Jurgen Schmidhuber, “Recurrent nets that time and count”, 2000. Διαθέσιμο στο: https://www.researchgate.net/publication/3857862_Recurrent_nets_that_time_and_count
16. Yoshua Bengio, Patrice Simard, Paolo Frasconi, “Learning long-term dependencies with gradient descent is difficult”, 1994. Διαθέσιμο στο: https://www.researchgate.net/publication/5583935_Learning_long-term_dependencies_with_gradient_descent_is_difficult
17. Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Fethi Bougares, Holger Schwenk, Yoshua Bengio, Dzmitry Bahdanau, “Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation”, 2014. Διαθέσιμο στο: https://www.researchgate.net/publication/262877889_Learning_Phrase_Representations_using_RNN_Encoder-Decoder_for_Statistical_Machine_Translation

18. Klaus Greff, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink, Jurgen Schmidhuber, “LSTM: A Search Space Odyssey”, 2015. Διαθέσιμο στο: https://arxiv.org/pdf/1503.04069.pdf
19. Rafal Jozefowicz, Wojciech Zaremba και Ilya Sutskever, “An Empirical Exploration of Recurrent Network Architectures”, 2015. Διαθέσιμο στο: https://proceedings.mlr.press/v37/jozefowicz15.pdf
20. www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
21. https://raw.githubusercontent.com/JDwangmo/papers_archive/master/NN_basic/Recurrent-Neural-Networks-Tutorial-Part-3%E2%80%93Backpropagation-Through-Time-and-Vanishing-Gradients%E2%80%93WildML.pdf
22. Denny Britz, 2015. Διαθέσιμο στο: https://www.kdnuggets.com/2015/10/recurrent-neural-networks-tutorial.html
23. Σημειώσεις του μαθήματος CS231n Convolutional Neural Networks for Visual Recognition, Stanford University. Διαθέσιμο στο: https://cs231n.github.io/neural-networks-1/
24. Herbert Jaeger, “A tutorial on training recurrent neural networks, covering BPPT, RTRL, EKF and the "echo state network" approach”, GMD Report 159, German National Research Center for Information Technology, 2002. Διαθέσιμο στο: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.4095&rep=rep1&type=pdf
25. Wikipedia, Conditional expectation, έκδοση της 4ης Οκτωβρίου 2021. Διαθέσιμο στο: https://en.wikipedia.org/wiki/Conditional_expectation
26. Devdarshan Mishra, “Applications of Recurrent Neural Networks (RNNs)”. Διαθέσιμο στο: https://iq.opengenus.org/applications-of-rnn/
27. “Unidecode 1.3.4”, έκδοση της 10ης Μαρτίου 2022. Διαθέσιμο στο: https://pypi.org/project/Unidecode/
28. Wikipedia, List of binary codes, έκδοση της 7ης Φεβρουαρίου 2022. Διαθέσιμο στο: https://en.wikipedia.org/wiki/List_of_binary_codes
29. Wikipedia, Braille, έκδοση της 17ης Ιουνίου 2022. Διαθέσιμο στο: https://en.wikipedia.org/wiki/Braille
30. “The Nemeth Braille code for Mathematics and Science notation”, 1972 Revision. Διαθέσιμο στο: https://nfb.org/images/nfb/documents/pdf/nemeth_1972.pdf
31. Wikipedia, TeX, έκδοση της 10ης Ιουνίου 2022. Διαθέσιμο στο: https://en.wikipedia.org/wiki/TeX
32. Andreas Papasalouros, Antonis Tsolomitis, “A direct TeX-to-Braille transcribing method”, Journal of Science Education for Students with Disabilities: Vol. 20: Iss. 1, pp. 36-49, Article 5, 2017. Διαθέσιμο στο: https://scholarworks.rit.edu/cgi/viewcontent.cgi?article=1071&context=jsesd