Αριστοτελειο Πανεπιστημιο Θεσσαλονικης

Τμημα Ηλεκτρολογων Μηχανικων και Μηχανικων Υπολογιστων

Τομεας Ηλεκτρονικης και Υπολογιστων

Προς την Αυτόματη Συγγραφή Κώδικα με Αναδραστικά Νευρωνικά Δίκτυα

Διπλωματικη Εργασια

του

ΒΑΣΙΛΗ ΜΠΟΥΝΤΡΗ

Επιβλέποντες: Ανδρέας Συμεωνίδης, Επίκουρος Καθηγητής Α.Π.Θ.
Κυριάκος Χατζηδημητρίου, Μεταδιδάκτορας Α.Π.Θ.

Εργαστηριο Επεξεργασιας Πληροφοριων και Υπολογισμων

Ομαδα Ευφυων Συστηματων και Τεχνολογιας Λογισμικου

Θεσσαλονίκη, Ιούνιος 2017

Ευχαριστίες

Θα ήθελα αρχικά να ευχαριστήσω τον επίκουρο καθηγητή κ. Ανδρέα Συμεωνίδη για την εμπιστοσύνη και την καθοδήγηση. Έπειτα θα ήθελα να ευχαριστήσω τον μεταδιδακτορικό ερευνητή κ. Κυριάκο Χατζηδημητρίου για την καθοδήγηση και τη συνεργασία. Τέλος θέλω να ευχαριστήσω την οικογένειά μου για την αδιάλειπτη στήριξη.

Περίληψη

Η εξέλιξη των κλάδων της Μηχανικής Μάθησης και της Επιστήμης της Πληροφορίας είναι ραγδαία την τελευταία δεκαετία. Ως μηχανικοί λογισμικού, αναζητούμε τρόπους να εκμεταλλευτούμε την εξέλιξη αυτή.

Στην παρούσα διπλωματική εργασία εξετάζουμε την ικανότητα των αναδραστικών νευρωνικών δικτύων στην παραγωγή κώδικα, ως δίκτυα αποτελεσματικά στη διαχείριση ακολουθιών. Προτείνουμε δύο προσεγγίσεις, που βασίζονται στην κατά χαρακτήρα ανάλυση αποθετηρίων κώδικα. Μετά την κατάλληλη προ-επεξεργασία του κώδικα και την εκπαίδευση των δικτύων, τα μοντέλα μπορούν να παράγουν κώδικα μέσω μιας στοχαστικής διαδικασίας. Εκτελούμε στατική ανάλυση κώδικα στα προϊόντα την μοντέλων, με σκοπό την εξέταση των επιδόσεων των προσεγγίσεων. Η γλώσσα επιλογής μας είναι η JavaScript.

Η ανάλυση δείχνει την μεγάλη αναπαραστατική δύναμη των αναδραστικών νευρωνικών δικτύων αλλά και την αδυναμία των προσεγγίσεων μας να αντιμετωπίσουν το πρόβλημα του αυτόματου προγραμματισμού. Με βάση αυτά τα ευρήματα προτείνουμε περαιτέρω ερευνητικές κατευθύνσεις και τρόπους εκμετάλλευσης των μοντέλων που σχεδιάστηκαν.

Λέξεις Κλειδιά

Αναδραστικά Νευρωνικά Δίκτυα, Παραγωγή Κώδικα, Χαρακτήρας, Ακολουθία, Μάθηση

Towards Source Code Generation with Recurrent Neural Networks

Abstract

The evolution of Machine Learning and Data Science disciplines has been rapid in the last decade. As software engineers, we are looking for ways to take advantage of this evolution.

In this diploma thesis we examine the ability of recurrent neural networks to generate code, because of their effectiveness at handling sequences. We propose two approaches, based on per-character analysis of software repositories. Following appropriate code preprocessing and network training, models can generate code through a stochastic process. We perform static code analysis on model products, in order to examine the performance of the approaches. The language we choose to work with is JavaScript.

The analysis shows the great representational power of the recurrent neural networks and the inability of our approaches to address the problem of automatic programming. Based on these findings, we propose further research directions and ways of exploiting the models that were designed.

Keywords

Recurrent Neural Networks, Source Code Generation, Character, Sequence, Learning

Vasilis Bountris
mvasilis@auth.gr
Aristotle University of Thessaloniki

June 2017

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

Η πράξη του προγραμματισμού, δηλαδή η ανάπτυξη μίας διαδικασίας με στόχο την επίτευξη ενός έργου, είναι μια εντυπωσιακή επίδειξη των δυνατοτήτων συλλογιστικής του ανθρώπινου εγκεφάλου. Η αυτοματοποίηση της συγγραφής κώδικα και προγραμμάτων (Αυτόματος Προγραμματισμός) είναι ένας στόχος με μακρόχρονη ιστορία, τόσο για τους μηχανικούς λογισμικού, όσο και για τον κλάδο της τεχνητής νοημοσύνης. Ο ακριβής ορισμός του "Αυτόματου Προγραμματισμού" παραμένει ένα θέμα στο οποίο υπάρχει ασυμφωνία μεταξύ των ειδικών, γεγονός που ενισχύεται από την συνεχή αλλαγή του όρου χάρη στις εξελίξεις της τεχνολογίας. Ο David Parnas, αναζητώντας την ιστορία του όρου, καταλήγει: "Ο αυτόματος προγραμματισμός ήταν πάντα ένας ευφημισμός για προγραμματισμό σε μια υψηλότερου επιπέδου γλώσσα από αυτή που είναι διαθέσιμη στον προγραμματιστή." [12]

Δεδομένης της εγγενούς δυσκολίας και πολυπλοκότητας του στόχου υπάρχει πληθώρα προκλήσεων αλλά και προσεγγίσεων στη λύση του. Δύο σημαντικές ομάδες προσεγγίσεων είναι [3], [14]:

1. Επαγωγικός Προγραμματισμός (Inductive Programming)

   Χρησιμοποιώντας τεχνικές τόσο από τον προγραμματισμό όσο και από την τεχνητή νοημοσύνη στοχεύει στη μάθηση προγραμμάτων, τυπικά δηλωτικών και συχνά αναδρομικών. Για την εκμάθηση χρησιμοποιούνται μη αυστηρές προδιαγραφές, όπως παραδείγματα εισόδου - εξόδου ή περιορισμοί.

2. Παραγωγή Κώδικα Βάσει Μοντέλων (Model-Driven Code Generation)

   Στην προσπάθεια των ερευνητών λογισμικού να απλοποιήσουν τη *διαδρομή* ανάμεσα στη σχεδίαση και την υλοποίηση, χρησιμοποιούν αφαιρέσεις. Ένα σύνολο τεχνικών με ευρεία και αυξανόμενη χρήση, το οποίο βασίζεται σε τέτοιες αφαιρέσεις, είναι το Model Driven Engineering. Γλώσσες μοντελοποίησης που αφορούν συγκεκριμένους τομείς (domain-specific modeling languages) χρησιμοποιούνται σε συνδυασμό με συστήματα μετατροπών και παραγωγής (transormation engines and generators) για να φτιάξουν τεχνουργήματα όπως προγράμματα, προσομοιώσεις εισόδου ή και άλλα μοντέλα.

Με ένα λειτουργικό σύστημα αυτόματης παραγωγής κώδικα, ο χρόνος ανάπτυξης και ο

αριθμός των λαθών μειώνεται δραματικά. Αντίστροφα, εκτινάσσεται η παραγωγικότητα των χρηστών και απλουστεύεται η αντιμετώπιση σύνθετων προβλημάτων.

1.1 Κίνητρο

Αφενός, η πρόοδος της τεχνητής νοημοσύνης, και ειδικότερα του κλάδου της μηχανικής μάθησης (machine learning), είναι ραγδαία τα τελευταία χρόνια. Αφετέρου, η εξέλιξη και η ευρεία χρήση του λογισμικού δημιουργεί ανάγκες για αυτοματοποίηση στην παραγωγή του. Έχουμε στη διάθεση μας πληθώρα υλοποιημένων προγραμμάτων, σε πολλές διαφορετικές γλώσσες και μορφές, για πληθώρα σκοπών. Οι σχετικές τεχνολογικές και θεωρητικές ανακαλύψεις ανοίγουν νέα μονοπάτια πειραματισμού, καινούρια εργαλεία αναπτύσσονται και δημιουργούνται κίνητρα επανεξέτασης κάποιων προβλημάτων.

Σύμφωνα με τα παραπάνω, και ιδιαίτερα χάρη στις πρόσφατες προόδους της μηχανικής μάθησης γύρω από την ταξινόμηση και την παραγωγή κειμένου [5], [9], καλούμαστε να εξετάσουμε πως και σε τι βαθμό μπορούμε να τις εκμεταλλευτούμε ως μηχανικοί λογισμικού. Τι εφαρμογές μπορούν να προκύψουν για την πρόβλεψη και τη διόρθωση κώδικα; Μέχρι ποιο σημείο μπορούμε να αυτοματοποιήσουμε την παραγωγή του;

1.2 Περιγραφή του προβλήματος

Το πρόβλημα που τίθεται προς λύση είναι η αυτοματοποίηση της παραγωγής κώδικα. Δεδομένων των σύγχρονων μεθόδων και τεχνολογιών, αυτό είναι ζήτημα στο οποίο είναι από εξαιρετικά δύσκολο έως αδύνατο να δοθεί μια γενική λύση, τουλάχιστον για το εγγύς μέλλον. Αντί για μία γενική λύση, μπορούμε να επικεντρωθούμε στις διεργασίες οι οποίες είναι μεν απλές, αλλά επαναλαμβάνονται συχνά και είναι χρονοβόρες. Ιδανικά, θα θέλαμε να αποφύγουμε να καταβάλουμε κόπο για να δημιουργήσουμε κάτι το οποίο ήδη υπάρχει.

1.3 Στόχοι της διπλωματικής

Στόχος της διπλωματικής εργασίας αυτής είναι η δημιουργία ενός τεχνητού νευρωνικού δικτύου με αναδράσεις (artificial recurrent neural network) το οποίο αφού εκπαιδευτεί στην συγγραφή κώδικα σε μία γλώσσα προγραμματισμού της επιλογής μας – διαβάζοντας εκατομμύρια γραμμές κώδικα – θα προσπαθήσει να παράξει κώδικα. Δεδομένου του εκπαιδευτικού χαρακτήρα της διπλωματικής εργασίας, θα εξετάσουμε το πρόβλημα αυτόματης παραγωγής κώδικα από μία πληροφοριακά οδηγούμενη (data-driven) σκοπιά, η οποία επιχειρεί να εκμεταλλευτεί τις εξελίξεις στην επιστήμη της πληροφορίας.

Ο κώδικας αυτός γενικά μπορεί να φτάσει σε ένα από τα παρακάτω επίπεδα:

1. Να "μοιάζει" με κώδικα
2. Να μην έχει συντακτικά λάθη

3. Να μπορεί να μεταφραστεί

4. Να "κάνει κάτι χρήσιμο"

Σε επίπεδο διπλωματικής εργασίας επιζητούμε κώδικα στα επίπεδα τουλάχιστον 1 ή και 2.

1.4 Μεθοδολογία

Θα αντιμετωπίσουμε την παραγωγή κώδικα ως ένα πρόβλημα εκμάθησης ακολουθιών (sequence learning), προσέγγιση η οποία βρίσκεται ανάμεσα στον επαγωγικό προγραμματισμό και την παραγωγή κώδικα βάσει μοντέλων. Χρησιμοποιούμε μοντέλα βασισμένα σε αναδραστικά νευρωνικά δίκτυα και ένα σύνολο δεδομένων. Το τελευταίο αποτελείται από έναν μεγάλο αριθμό προγραμμάτων σε μια γλώσσα της επιλογής μας. Σε αυτή την περίπτωση θα χρησιμοποιήσουμε τη γλώσσα JavaScript, αλλά το μοντέλο μας είναι αγνωστικό στο ποια γλώσσα μαθαίνει. Η μεθοδολογία μπορεί να χωριστεί, αφαιρετικά, σε 3 μέρη:

1. Προ-επεξεργασία (Pre-processing)

   Δεδομένου ενός μεγάλου όγκου πληροφοριών σε μορφή κώδικα, καλούμαστε να τις επεξεργαστούμε με στόχο την καλύτερη εκμετάλλευση τους από το μοντέλο μας και τελικώς την επίτευξη βέλτιστων αποτελεσμάτων. Αφαιρούμε την πληροφορία που φαίνεται είτε να δυσκολεύει την εκμάθηση του μοντέλου, είτε είναι αδύνατο να ερμηνευτεί από αυτό. Σε μία από τις προτεινόμενες προσεγγίσεις προσθέτουμε πληροφορία για τον κώδικα με σκοπό την αποσαφήνιση των δεδομένων. Η πληροφορία του κώδικα εκφράζεται ως σειρά από στοιχειώδεις χαρακτήρες.

2. Εκπαίδευση (Training)

   Τα προτεινόμενα μοντέλα, τα οποία είναι σύνθετες δομές αναδραστικών νευρωνικών δικτύων, εκπαιδεύονται βάσει της παραπάνω επεξεργασμένης πληροφορίας. Μετά από το *διάβασμα* μιας σειράς χαρακτήρων καλούνται να προβλέψουν τον επόμενο χαρακτήρα. Οι επιδόσεις εκφράζονται μέσω μιας μετρικής λάθους, την οποία η εκπαιδευτική διαδικασία προσπαθεί να ελαχιστοποιήσει χρησιμοποιώντας γενικευμένες μεθόδους βελτιστοποίησης.

3. Παραγωγή Κώδικα (Source Code Generation)

   Τα εκπαιδευμένα, πλέον, μοντέλα μπορούν να χρησιμοποιηθούν για την παραγωγή κώδικα. Αρχικοποιούνται με κώδικα της επιλογής μας, για τον οποίο ακολουθούμε την ίδια μέθοδο προ-επεξεργασίας με αυτή των δεδομένων εκπαίδευσης. Το μοντέλο παράγει ένα χαρακτήρα σε κάθε πρόβλεψη και χρησιμοποιεί την πρόβλεψη του ως αληθή για να παράξει τον επόμενο χαρακτήρα. Με αυτό τον τρόπο μπορεί να συγγράφει απεριόριστη ποσότητα κώδικα.

1.5 Διάρθρωση

Η εργασία αυτή είναι οργανωμένη σε έξι κεφάλαια: Στο Κεφάλαιο 2 δίνεται το θεωρητικό υπόβαθρο των βασικών τεχνολογιών που σχετίζονται με τη διπλωματική αυτή. Στο κεφάλαιο 3 παρουσιάζονται σχετικές υλοποιήσεις συστημάτων μηχανικής μάθησης και αυτόματου προγραμματισμού. Στο Κεφάλαιο 4 παρουσιάζεται και αναλύεται η μεθοδολογία που χρησιμοποιείται, δηλαδή τα επιμέρους μοντέλα και οι μέθοδοι εκπαίδευσης και παραγωγής. Στο Κεφάλαιο 5 δίνονται αναλυτικά πληροφορίες για τα πειράματα και τα αποτελέσματα τους. Τέλος, στο Κεφάλαιο 6 συζητούνται τα συμπεράσματα καθώς και μελλοντικές επεκτάσεις της διπλωματικής εργασίας.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά οι τεχνολογίες που είναι απαραίτητες για την κατανόηση της διπλωματικής εργασίας.

2.1 Βαθιά Μάθηση

Η μηχανική μάθηση είναι η κινητήριος δύναμη για διάφορες εκφάνσεις της σύγχρονης κοινωνίας: από αναζητήσεις στο διαδίκτυο μέχρι το φιλτράρισμα περιεχομένου σε κοινωνικά δίκτυα και προτάσεις αγορών σε ηλεκτρονικά καταστήματα. Ολοένα συχνότερη και συνηθέστερη γίνεται η εμφάνιση της σε προϊόντα ευρείας κατανάλωσης όπως κάμερες και κινητά τηλέφωνα. Τα συστήματα μηχανικής μάθησης χρησιμοποιούνται για την αναγνώριση αντικειμένων σε εικόνες, την αυτόματη καταγραφή προφορικού λόγου, την αντιστοίχηση προϊόντων - νέων - δημοσιεύσεων με τις προτιμήσεις χρηστών. Σε όλες αυτές τις εφαρμογές, είναι αυξανόμενη η χρήση ενός σετ τεχνικών που φέρει το όνομα Βαθιά Μάθηση (Deep Learning) [8].

Οι συμβατικές τεχνικές υπολογιστικής εκμάθησης είχαν περιορισμένη δυνατότητα χρήσης της ανεπεξέργαστης πληροφορίας. Για δεκαετίες, η σχεδίαση και η υλοποίηση ενός συστήματος αναγνώρισης προτύπων ή υπολογιστικής εκμάθησης, απαιτούσε προσεκτική προσέγγιση και σημαντική εξειδίκευση στον εκάστοτε τομέα. Αυτό επειδή χρειαζόταν η μετατροπή της ανεπεξέργαστης πληροφορίας σε μία κατάλληλη εσωτερική αναπαράσταση, την οποία το υποσύστημα εκμάθησης – συχνότερα ένας ταξινομητής – θα χρησιμοποιούσε για αναγνωρίσει πρότυπα στις διάφορες εισόδους.

Η εκμάθηση αναπαραστάσεων είναι ένα σύνολο μεθόδων που επιτρέπουν σε ένα σύστημα να ανακαλύψει αυτόματα ποιες ακριβώς αναπαραστάσεις της ανεπεξέργαστης πληροφορίας χρειάζεται, για να επιτελέσει την αναγνώριση προτύπων ή την ταξινόμηση. Οι μέθοδοι βαθιάς μάθησης είναι μέθοδοι εκμάθησης αναπαραστάσεων με πολλαπλά επίπεδα αναπαράστασης, που αποτελούνται από την σύνθεση απλών, μη γραμμικών υποσυστημάτων, το καθένα από τα οποία – ξεκινώντας από την ανεπεξέργαστη είσοδο – μετατρέπει την αναπαράσταση της πληροφορίας σε μια λίγο πιο υψηλά αφαιρετική μορφή σε κάθε επίπεδο. Με την χρήση αρκετών τέτοιων μετατροπών το σύστημα μπορεί να μάθει εξαιρετικά σύνθετες λειτουργίες. Για διαδικασίες

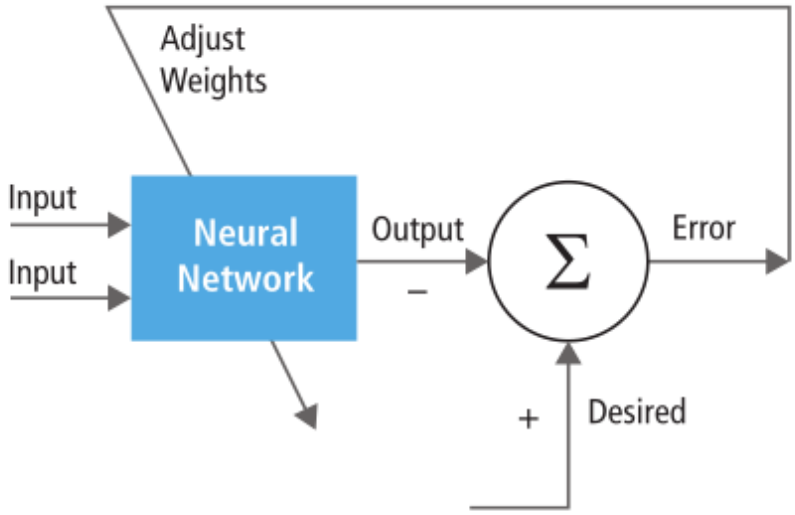
ταξινόμησης, τα υψηλότερα επίπεδα αναπαράστασης ενισχύουν πτυχές τις εισόδου που είναι πιο σημαντικές για τον τελικό σκοπό. Σε μία εικόνα, για παράδειγμα, η οποία αναπαριστάται ως διάνυσμα τιμών εικονοκυττάρων, τα χαρακτηριστικά που μαθαίνονται στο πρώτο επίπεδο είναι συνήθως πληροφορία για την παρουσία ή την απουσία ακμών σε συγκεκριμένες θέσεις και προσανατολισμούς. Στο δεύτερο επίπεδο, συνήθως εντοπίζονται μοτίβα μέσω των διαφόρων διατάξεων των ακμών, χωρίς να χρειάζεται τα μοτίβα να επαναλαμβάνονται επακριβώς. Στο τρίτο επίπεδο μπορούν να αναγνωριστούν σύνολα μοτίβων σε μεγάλους συνδυασμούς που αντιστοιχούν σε γνωστά αντικείμενα ή μέρη τους. Τα επόμενα επίπεδα, παρόμοια, εντοπίζουν πιο σύνθετα αντικείμενα ως συνδυασμούς απλούστερων μερών. Το βασικότερο στοιχείο της βαθιάς μάθησης είναι πως τα επίπεδα που εντοπίζουν χαρακτηριστικά και δομές δεν είναι σχεδιασμένα από τους ανθρώπους: μαθαίνονται από τα δεδομένα χρησιμοποιώντας γενικευμένες διαδικασίες εκμάθησης.

Η χρήση της έχει βοηθήσει στην αντιμετώπιση προβλημάτων που δυσκόλευαν την κοινότητα της τεχνητής νοημοσύνης εδώ και χρόνια. Παρατηρείται να έχει επιδόσεις χωρίς προηγούμενο στον εντοπισμό πολύπλοκων δομών σε δεδομένα πολλών διαστάσεων και για αυτό είναι εφαρμόσιμο σε πολλούς διαφορετικούς τομείς, επιστημονικούς, επιχειρησιακούς και κοινωνικοπολιτικούς. Πέρα από επαναστατικές επιδόσεις στην αναγνώριση φωνής και εικόνας, έχει ξεπεράσει άλλες τεχνικές υπολογιστικής εκμάθησης στην πρόβλεψη συμπεριφοράς μορίων φαρμάκων, στην ανάλυση δεδομένων από επιταχυντές σωματιδίων, στην ανακατασκευή εγκεφαλικών κυκλωμάτων και στην πρόβλεψη των επιπτώσεων μεταλλάξεων μη κωδικοποιητικού DNA στις γονιδιακές εκφράσεις και ασθένειες. Ίσως, οι πιο αναπάντεχα υποσχόμενες επιδόσεις έγιναν στον κλάδο της επεξεργασίας φυσικής γλώσσας, συγκεκριμένα στην εντοπισμό θεμάτων, την ανάλυση συναισθήματος, τα συστήματα ερωταπαντήσεων και την μετάφραση.

2.2 Επιτηρούμενη Μάθηση

Η πιο συνήθης μορφή μηχανικής μάθησης είναι αυτή της επιτηρούμενης μάθησης (supervised learning). Ας θεωρήσουμε πως θέλουμε να φτιάξουμε ένα σύστημα που αποφασίζει τι περιέχει μια εικόνα, όπως ένα σπίτι, ένα αυτοκίνητο, έναν άνθρωπο ή μία γάτα. Αρχικά, συλλέγουμε ένα αρκετά μεγάλο σύνολο δεδομένων με εικόνες στα οποία σημειώνεται τι αντικείμενο από τα παραπάνω περιέχει κάθε εικόνα. Κατά τη διάρκεια της εκπαίδευσης, δείχνουμε στο σύστημα μια εικόνα και αυτό παράγει μία πρόβλεψη, στη μορφή ενός διανύσματος με σκορ για κάθε κατηγορία. Θέλουμε η επιθυμητή κατηγορία να έχει το μεγαλύτερο σκορ πρόβλεψης, αλλά αυτό είναι πολύ δύσκολο πριν την εκπαίδευση. Υπολογίζουμε μία συνάρτηση στόχου με την οποία μετράμε το λάθος (ή την απόσταση) μεταξύ των αποτελεσμάτων του συστήματος και τον επιθυμητών αποτελεσμάτων. Το σύστημα, ύστερα, προσαρμόζει τις εσωτερικές του παραμέτρους ώστε να μειώσει το λάθος. Οι εσωτερικές παράμετροι, που συχνότερα στη βιβλιογραφία συναντώνται ως βάρη, είναι πραγματικοί αριθμοί που ορίζουν την λειτουργικότητα εισόδου-εξόδου του συστήματος. Σε ένα τυπικό σύστημα βαθιάς μάθησης, οι εσωτερικές παράμετροι και τα παραδείγματα που χρησιμοποιούμε για την εκμάθηση του συστήματος μπορεί να είναι εκατοντάδες εκατομμύρια σε αριθμό.

Για την κατάλληλη προσαρμογή των βαρών, ο αλγόριθμος εκμάθησης υπολογίζει ένα διάνυσμα κλίσης, για κάθε βάρος, που δείχνει κατά πόσο και προς ποια κατεύθυνση αλλάζει το λάθος αν αλλάξουμε απειροστά το αντίστοιχο βάρος. Το διάνυσμα των βαρών τελικά ρυθμίζεται έτσι ώστε να έχει αντίθετη φορά με το διάνυσμα κλίσης. Η διαδικασία αυτή είναι μία προσπάθεια ελαχιστοποίησης της συνάρτησης λάθους και μεταγενέστερα της μείωσης, κατά μέσο όρο, των λαθών προβλέψεων του συστήματος.



Σχήμα 2.1: Ένα τυπικό δομικό διάγραμμα επιτηρούμενης μάθησης.

Στην πλειοψηφία της σύγχρονης βιβλιογραφίας, και στην παρούσα διπλωματική, ο αλγόριθμος ελαχιστοποίησης που χρησιμοποιείται είναι ο stochastic gradient descent (SGD). Αυτός συνίσταται από την επίδειξη λίγων κάθε φορά, σωστά επισημασμένων, παραδειγμάτων στο σύστημα, τον υπολογισμό των προβλέψεων και του λάθους, τον υπολογισμό του διανύσματος κλίσης και την ρύθμιση των βαρών. Η παραπάνω διαδικασία επαναλαμβάνεται για πολλά μικρά σετ παραδειγμάτων, μέχρι η συνάρτηση στόχου να σταματήσει να μειώνεται. Μετά την εκπαίδευση, οι επιδόσεις του συστήματος μετρώνται σε ένα σύνολο διαφορετικών παραδειγμάτων, έτσι ώστε να εξεταστεί η ικανότητα γενίκευσης του συστήματος σε εισόδους που βλέπει για πρώτη φορά.

2.3 Αναδραστικά Νευρωνικά Δίκτυα

Τα αναδραστικά νευρωνικά δίκτυα (Recurrent Neural Networks - RNNs) είναι μία προσαρμογή των κλασσικών, πλήρως συνδεδεμένων νευρωνικών δικτύων, έτσι ώστε τα πρώτα να μπορούν να διαχειριστούν ακολουθίες. Σε κάθε χρονική στιγμή, τα RNNs δέχονται μια είσοδο, ενημερώνουν την εσωτερική τους κατάσταση και παράγουν μία έξοδο. Η πολυδιάστατη εσωτερική κατάσταση, που συχνά απαντάται στη βιβλιογραφία ως κρυφή κατάσταση (hidden state), και η μη γραμμική εξέλιξη της διαχειριζόμενης πληροφορίας δίνουν στα αναδραστικά νευρωνικά δίκτυα μεγάλη εκφραστική ευελιξία και δυνατότητα ενσωμάτωσης και διατήρησης

της πληροφορίας σε μεγάλα χρονικά διαστήματα. Ακόμα και όταν η μη γραμμική συνάρτηση που χρησιμοποιείται από κάθε στοιχείο του RNN είναι εξαιρετικά απλή, η χρήση της σε πολλά επίπεδα και η επανάληψη της σε κάθε χρονική στιγμή οδηγεί σε ένα εξαιρετικά δυναμικό σύστημα.

Τα αναδραστικά νευρωνικά δίκτυα ορίζονται ως εξής: δεδομένης μιας ακολουθίας διανυσμάτων εισόδου $(x_1, x_2, ..., x_T)$, το σύστημα υπολογίζει μία ακολουθία κρυφών καταστάσεων $(h_1, h_2, ..., h_T)$ και μία παράγει μια ακολουθία εξόδων $(o_1, o_2, ..., o_T)$, σύμφωνα με τον κάτωθι αλγόριθμο:

Algorithm 1 RNN

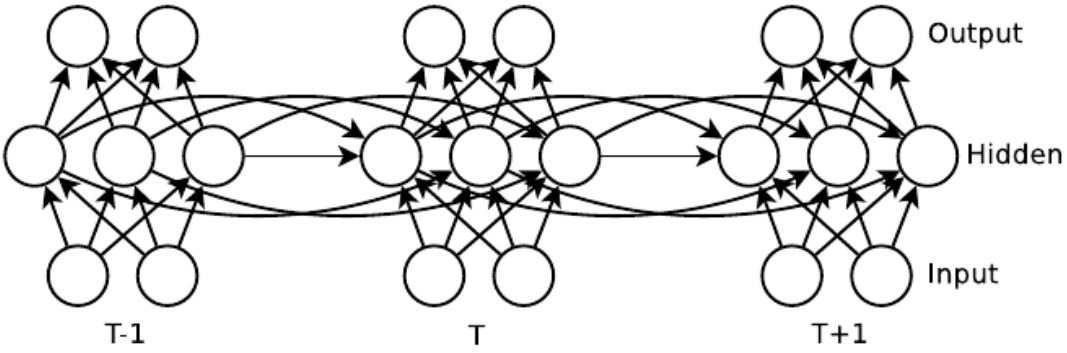
for $t = 1$ to T **do**

$$h_t = F(W_h x_t + W_{hh} h_{t-1} + b_h) \tag{2.1}$$

$$o_t = W_{oh} h_t + b_o \tag{2.2}$$

end for

Σε αυτές τις εξισώσεις, το W_{hx} είναι ο πίνακας βαρών από την είσοδο στην κρυφή κατάσταση, το W_{hh} είναι ο πίνακας απο την κρυφή κατάσταση στην κρυφή κατάσταση, το W_{oh} είναι ο πίνακας βαρών από την κρυφή κατάσταση στην έξοδο και τα b_h, b_o είναι οι σταθεροί όροι. Η μη ορισμένη σχέση $W_{hh} h_{t-1}$ στη χρονική στιγμή $t = 1$ αντικαθίσταται με ένα διάνυσμα αρχικοποίησης, h_{init}, και η συνάρτηση F είναι μία συνάρτηση ενεργοποίησης των στοιχείων που εφαρμόζεται κατά στοιχείο.



Σχήμα 2.2: Το αναδραστικό νευρωνικό δίκτυο ανεπτυγμένο στον χρόνο

Οι παράγωγοι των στοιχειδών μερών του δικτύου είναι εύκολο να υπολογιστούν, με τη μέθοδο της προς τα πίσω διάδοσης σφάλματος, [5], οπότε η εκπαίδευση ενός τέτοιου συστήματος φαίνεται εύκολη. Στην πραγματικότητα, η σχέση μεταξύ των παραμέτρων του RNN και της δυναμικής του είναι εξαιρετικά ασταθής, γεγονός που καθιστά τον αλγόριθμο SGD αναποτελεσματικό. Αυτό τεκμηριώνεται απο τους Pascanu et al. [13] που αποδεικνύουν πως τα διανύσματα κλίσεων τείνουν να μηδενίζονται (ή σπανιότερα να απειρίζονται) εκθετικά με την διάδοση του σφάλματος στο χρόνο. Στη σχετική βιβλιογραφία αυτό συναντάται ως πρόβλημα εξαφάνισης ή έκρηξης των διανυσμάτων κλίσης ("vanishing or exploding gradients

problem"). Το παραπάνω χρησιμοποιήθηκε ως επιχείρημα για το ότι τα αναδραστικά νευρωνικά δίκτυα δεν μπορούν να αποτυπώσουν εξαρτήσεις με μεγάλη χρονική απόσταση μεταξύ τους, όταν ο χρησιμοποιείται ο αλγόριθμος SGD. Επιπρόσθετα, ο περιστασιακός απειρισμός των διανυσμάτων κλίσης αυξάνει τη διακύμανση τους και κάνει την εκμάθηση ασταθή. Τα θεωρητικά αποτελέσματα αυτά, δεδομένου πως ο SGD ήταν ο βασικότερος αλγόριθμος εκπαίδευσης νευρωνικών δικτύων, σε συνδυασμό με την εμπειρική δυσκολία εκπαίδευσης των RNNs οδήγησε στη σχεδόν ολοκληρωτική εγκατάλειψη της σχετικής έρευνας.

2.4 Εκπαίδευση των Αναδραστικών Νευρωνικών Δικτύων

2.4.1 Long Short-Term Memory Units

Ένας τρόπος να αντιμετωπιστεί η αδυναμία που παρουσιάζουν τα RNNs στην εκμάθησης δομών με μακρινές, στο χρόνο, αλληλεξαρτήσεις είναι η τροποποίηση του μοντέλου ώστε να έχει στοιχεία με μνήμη. Η προσέγγιση αυτή ονομάζεται Long Short-Term Memory (LSTM)[6] και γνωρίζει ευρεία χρήση. Οι σχέσεις που ορίζουν κάθε στοιχείο μνήμης είναι:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \tag{2.3}$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \tag{2.4}$$

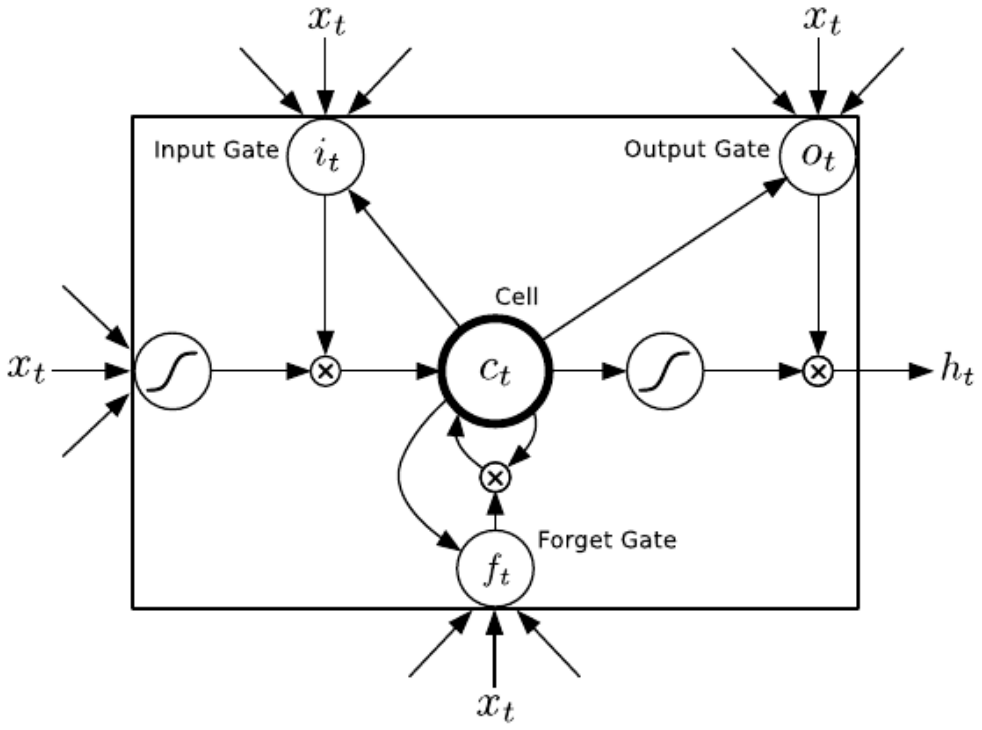
$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.5}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \tag{2.6}$$

$$h_t = o_t \tanh c_t \tag{2.7}$$

Το σ είναι η σιγμοειδής συνάρτηση, i, f, o και c είναι αντίστοιχα η πύλη εισόδου, η πύλη απώλειας μνήμης, η πύλη εξόδου και η μνήμη. Τα τελευταία είναι διανύσματα με διαστάσεις ίδιες με του διανύσματος h (βλ. εξίσωση 2.1, που ταυτίζεται με τον δεύτερο όρο της εξίσωσης 2.5). Ο W_{hi} είναι ο πίνακας βαρών προηγούμενης κρυφής κατάστασης – πύλης εισόδου, ο πίνακας W_{ho} είναι ο πίνακας βαρών προηγούμενης κατάστασης – πύλης εξόδου, ο πίνακας W_{hf} είναι ο πίνακας βαρών προηγούμενης κατάστασης – πύλης απώλειας μνήμης, ο πίνακας W_{hc} είναι ο πίνακας βαρών προηγούμενης κατάστασης – μνήμης, ο W_{xi} είναι πίνακας βαρών εισόδου – πύλης εισόδου, ο W_{xo} είναι πίνακας βαρών εισόδου – πύλης εξόδου, ο W_{xf} είναι ο πίνακας βαρών εισόδου – πύλης απώλειας μνήμης, ο W_{xc} είναι πίνακας βαρών εισόδου – μνήμης, ο W_{ci} είναι ο πίνακας βαρών μνήμης – πύλης εισόδου, ο W_{cf} είναι ο πίνακας βαρών μνήμης – πύλης απώλειας μνήμης, ο W_{co} είναι ο πίνακας βαρών μνήμης – πύλης εξόδου. Οι πίνακες βαρών από την μνήμη στις πύλες είναι διαγώνιοι, έτσι το στοιχείο m σε κάθε πύλη δέχεται είσοδο μόνο από το στοιχείο m του διανύσματος μνήμης. Στο σχήμα 2.3 απεικονίζεται το σύστημα LSTM.

Μια πιο διαισθητική εξήγηση του συστήματος LSTM είναι η εξής: Η μνήμη c, σε κάθε επανάληψη της λειτουργίας του αναδραστικού νευρωνικού δικτύου, αλλάζει δυναμικά. Μέσω



Σχήμα 2.3: Το σύστημα LSTM

της πύλης απώλειας μνήμης f αρχικά αποφασίζεται πιο κομμάτι της υπάρχουσας πληροφορίας της c θα κρατήσουμε "κοιτώντας" την είσοδο x_t και την προηγούμενη κατάσταση h_{t-1}. Ύστερα η πύλη εισόδου αποφασίζει πιο κομμάτι της εισόδου θα αποθηκευτεί. Αποθηκεύεται η καινούρια μνήμη συνδυάζοντας τις αποφάσεις τον προηγούμενων βημάτων. Τέλος η πύλη εξόδου αποφασίζει πιο κομμάτι της μνήμης θα εξαχθεί.

2.4.2 Truncated Backpropagaion Through Time

Ένα από τα βασικά προβλήματα του αλγορίθμου της προς τα πίσω διάδοσης του σφάλματος, είναι το υψηλό κόστος για την ενημέρωση μιας μεμονωμένης παραμέτρου, γεγονός που την καθιστά απαγορευτική για πολλές επαναλήψεις. Για παράδειγμα, ο υπολογισμός του διανύσματος κλίσεων ενός RNN ακολουθιών μήκους 1000 στοιχειών, στοιχίζει όσο και το εμπρόσθιο και προς τα πίσω πέρασμα ενός πλήρως συνδεδεμένου νευρωνικού δικτύου 1000 επιπέδων. Το υπολογιστικό κόστος μπορεί να μειωθεί με μία μέθοδο που χωρίζει την ακολουθία 1000 στοιχείων σε, για παράδειγμα, 50 ακολουθίες μήκους 20 στοιχείων η καθεμία και τις αντιμετωπίζει ως ξεχωριστά παραδείγματα εκπαίδευσης. Αυτή η απλή προσέγγιση μπορεί να εκπαιδεύσει το νευρωνικό δίκτυο ικανοποιητικά, αλλά αδυνατεί να αποτυπώσει σχέσεις που εκτείνονται παραπάνω από 20 χρονικές στιγμές. Ο αλγόριθμος Truncated Bacpropagation Through Time είναι μία συναφής μέθοδος. Έχει το ίδιο κόστος με την απλή μέθοδο που

περιγράψαμε παραπάνω αλλά είναι πιο ικανός στο να αποτυπώνει χρονικές εξαρτήσεις μεγάλου μήκους. Επεξεργάζεται την ακολουθία ένα στοιχείο τη φορά, και κάθε k_1 στοιχεία, καλεί τον αλγόριθμο BPTT για k_2 στοιχεία, έτσι η ενημέρωση των παραμέτρων είναι πιο φθηνή επεξεργαστικά αν το k_2 είναι αρκούντως μικρό. Συνεπώς, η κρυφή κατάσταση εκτίθεται σε πολλά στοιχεία και μπορεί να περιέχει χρήσιμη πληροφορία για το παρελθόν της ακολουθίας γεγονός το οποίο μπορούμε να εκμεταλλευτούμε. Ο αλγόριθμος Truncated Backpropagation Through Time:

Algorithm 2 Truncated Backpropagation Through Time

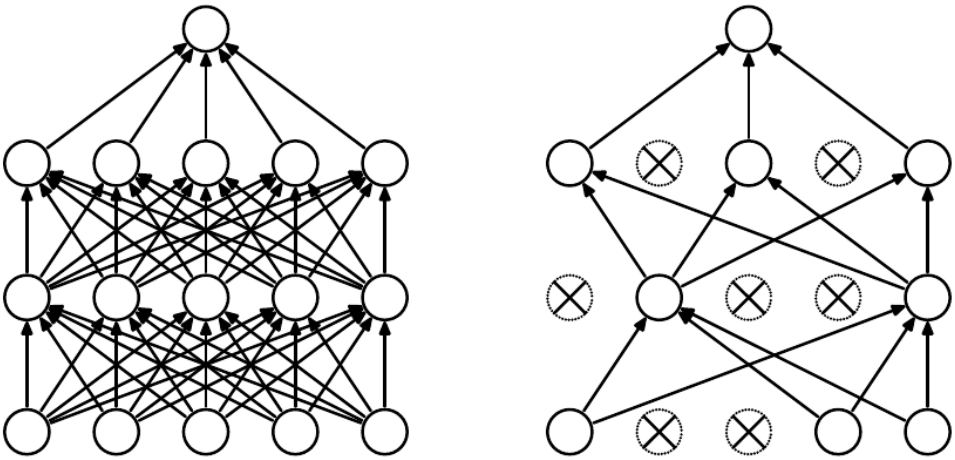
```
for t = 1 to T do
  RNN iteration
  if t divides k1 then
    BPPT from t to t − k2
  end if
end for
```

2.4.3 Dropout

Τα βαθιά νευρωνικά δίκτυα περιέχουν πολλαπλά μη γραμμικά επίπεδα γεγονός αυτό τα κάνει εξαιρετικά εκφραστικά μοντέλα ώστε να μπορούν να μάθουν περίπλοκες σχέσεις μεταξύ εισόδου και εξόδου. Με περιορισμένα όμως δεδομένα εκπαίδευσης, πολλές από τις σχέσεις που αποτυπώνονται μπορεί να είναι αποτέλεσμα θορύβου δειγματοληψίας. Έτσι, οι επιδόσεις των μοντέλων στα δεδομένα εκπαίδευσης μπορεί να διαφέρουν σημαντικά από τις επιδόσεις στα πραγματικά δεδομένα. Αυτό οδηγεί στο overfitting και διάφοροι μέθοδοι έχουν αναπτυχθεί για την αντιμετώπισή του, όπως οι L1 και L2 κανονικοποιήσεις.

Μία τέτοια τεχνική κανονικοποίησης είναι και το Dropout [15]. Συνοπτικά, μας δίνει τη δυνατότητα να συνδυάσουμε προσεγγιστικά εκθετικά πολλές διαφορετικές αρχιτεκτονικές νευρωνικών δικτύων, αποτελεσματικά. Ο όρος dropout (απόσυρση), αναφέρεται στην παράλειψη στοιχείων του νευρωνικού δικτύου. Παραλείποντας ένα στοιχείο εννοούμε την προσωρινή του αφαίρεση από το δίκτυο, μαζί με τις συνδέσεις από και προς αυτό. Η επιλογή των στοιχείων που παραλείπονται είναι τυχαία. Στην πιο απλή εφαρμογή, κάθε στοιχείο κρατείται με μία πιθανότητα p που είναι σταθερή και ανεξάρτητη των υπολοίπων στοιχείων.

Η χρήση της τεχνικής dropout αναλογεί στη χρήση διαφόρων "αραιωμένων" δικτύων που βασίζονται στο αρχικό. Το αραιωμένο δίκτυο αποτελείται από τα στοιχεία που "επέζησαν" της χρήσης dropout (Σχήμα 2.4) και για κάθε παράδειγμα από το σετ εκπαίδευσης επιλέγεται τυχαία ένα αραιωμένο δίκτυο. Έτσι, νευρωνικό δίκτυο με n στοιχεία μπορεί να θεωρηθεί μια συλλογή από 2^n πιθανά αραιωμένα νευρωνικά δίκτυα τα οποία μοιράζονται βάρη με το αρχικό, και η εκπαίδευση του αρχικού ανάγεται στην εκπαίδευση της συλλογής αραιωμένων δικτύων. Για την εκτίμηση των επιδόσεων του συστήματος χρησιμοποιούμε όλα τα στοιχεία του νευρωνικού δικτύου, αλλά τα εξερχόμενα βάρη τους είναι πολλαπλασιασμένα με την πιθανότητα p.



Σχήμα 2.4: Ένα νευρωνικό δίκτυο με dropout (δεξιά) και χωρίς (αριστερά).

Κεφάλαιο 3

Σχετική βιβλιογραφία

Στο κεφάλαιο αυτό παρουσιάζουμε σχετικές προσεγγίσεις και υλοποιήσεις στο πρόβλημα του αυτόματου προγραμματισμού και της παραγωγής κώδικα. Επειδή είναι πρακτικά αναρίθμητες, θα επικεντρωθούμε σε αυτές που είναι σχετικές με τα αναδραστικά νευρωνικά δίκτυα και σε κάποιες που παρουσιάζουν ιδιαίτερο ενδιαφέρον.

3.1 Generating Sequence with Recurrent Neural Networks

Στην εργασία των Graves et al. [5] παρουσιάζεται πως απλές δομές αναδραστικών νευρωνικών δικτύων με στοιχεία μνήμης LSTM μπορούν να χρησιμοποιηθούν για να παράξουν σύνθετες ακολουθίες, απλά προβλέποντας ένα στοιχείο της ακολουθίας τη φορά. Δεδομένου ενός εκπαιδευμένου δικτύου, μπορούμε να κάνουμε δειγματοληψία στην έξοδο του και την ξαναδώσουμε ως είσοδο. Έτσι δημιουργούνται καινούριες ακολουθίες. Αν και το σύστημα είναι ντετερμινιστικό, η στοχαστικότητα που εισάγεται δειγματοληπτώντας δημιουργεί μία κατανομή σε σχέση με τις ακολουθίες. Αυτή η κατανομή είναι δεσμευμένη, αφού η εσωτερική αναπαράσταση του δικτύου, άρα και κατανομή προβλέψεων του, εξαρτάται από τις προηγούμενες εισόδους.

Η προσέγγιση αυτή επιδεικνύεται για δημιουργία κείμενου (όπου οι τιμές είναι διακριτές) και για "online" παραγωγή χειρόγραφου κείμενου (όπου οι τιμές είναι πραγματικές). Με τον όρο "online" εννοούμε ότι η γραφή αποτυπώνεται ως ακολουθία διανυσμάτων θέσης ενός μολυβιού – σε αντίθεση με το "offline" στο οποίο έχουμε διαθέσιμη ολόκληρη την εικόνα του χειρόγραφου. Το σύστημα που χρησιμοποιείται είναι μια συστάδα που αποτελείται από 7 επίπεδα αναδραστικών νευρωνικών δικτύων με 700 στοιχεία μνήμης LSTM. Για την παραγωγή ακολουθιών κειμένου χρησιμοποιούνται τρία διαφορετικά σετ δεδομένων. Το Penn Treebank[1] και το Text8[2] για το γραπτό κείμενο και το IAM online handwriting database[3]. Το μοντέλο καταφέρνει να παράξει ακολουθίες τόσο ρεαλιστικές ώστε να είναι συχνά δύσκολο να τις ξεχωρίσει κανείς από πραγματικές, τουλάχιστον σε πρώτη όψη. Στα αποτελέσματα είναι ορατή

[1] `https://catalog.ldc.upenn.edu/ldc99t42`

[2] `http://mattmahoney.net/dc/textdata`

[3] `http://www.fki.inf.unibe.ch/databases/iam-on-line-handwriting-database`

μια μεγάλης εμβέλειας δομή και συνοχή.

Επιπρόσθετα, βασισμένοι στην προηγούμενη δομή, οι Graves et al. δημιουργούν ένα σύστημα παραγωγής χειρόγραφου κειμένου το οποίο μπορεί να γράψει αυτό που του ζητάμε. Αυτό γίνεται με την προσθήκη ενός διανύσματος της πρότασης που θέλουμε να γράψουμε, το οποίο δίνεται στο σύστημα πρόβλεψης την ώρα της παραγωγής, αφού προφανώς εκπαιδευτεί σε σχετικά προβλήματα. Η απόφαση για το πότε και πως θα γραφεί κάθε χαρακτήρας αφήνεται στο νευρωνικό δίκτυο και τα αποτελέσματα είναι αρκετά ικανοποιητικά ώστε να είναι και πάλι δύσκολο να διακριθεί αν τα "χειρόγραφα" ανήκουν σε κάποιον άνθρωπο ή στο σύστημα. Στο σχήμα 3.1 φαίνονται 2 παραδείγματα του συστήματος. Η πρώτη γραμμή σε κάθε παράδειγμα είναι γραμμένη από έναν άνθρωπο και οι υπόλοιπες είναι προσπάθειες του συστήματος να συγγράψει την πρώτη γραμμή.

3.2 Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets

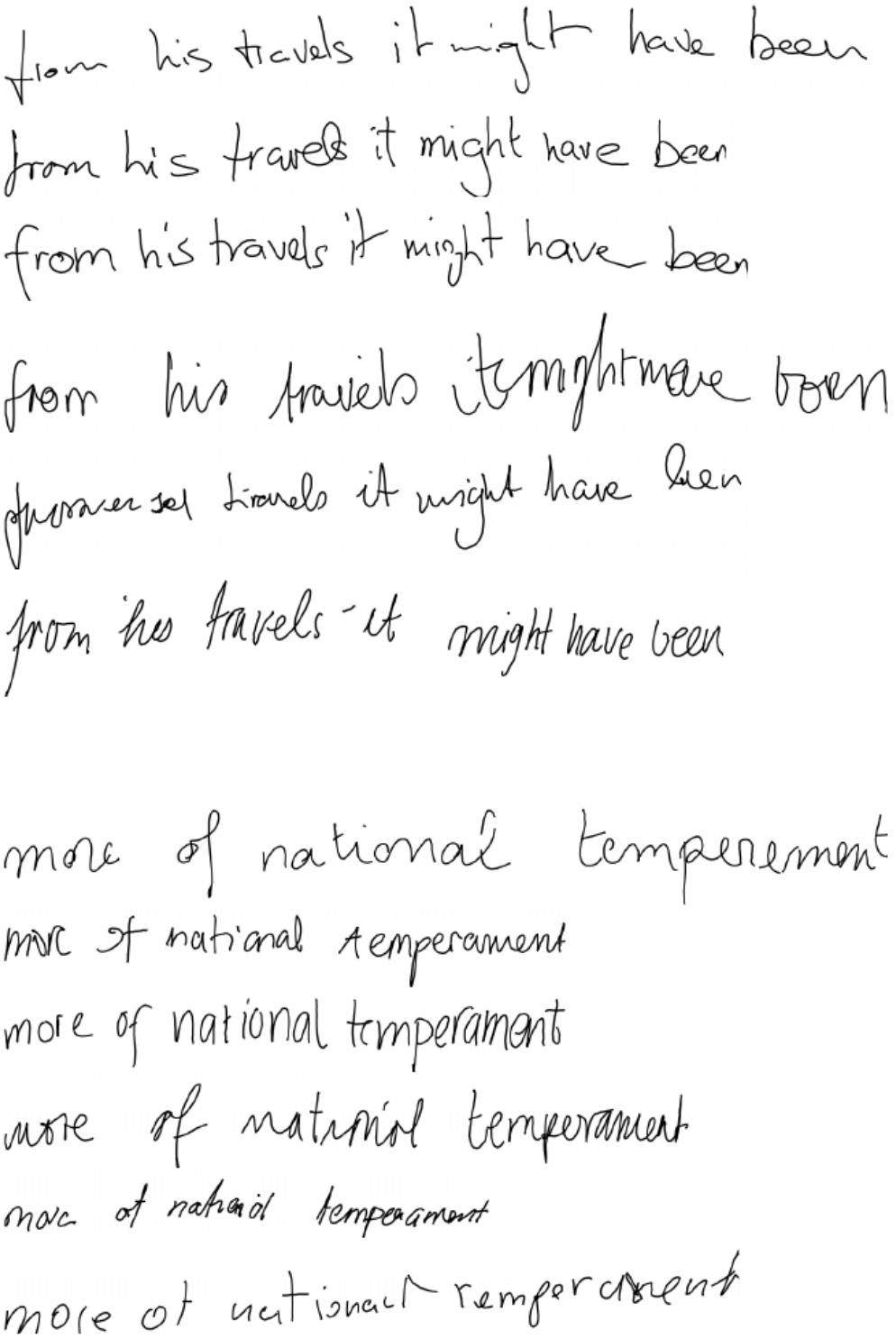
Οι Joulin et al. [7] στην έρευνα τους εξετάζουν τα όρια των "state of the art" Deep Learning προσεγγίσεων. Πιο συγκεκριμένα, εξετάζονται τα απλούστερα προβλήματα πρόβλεψης ακολουθιών που είναι πέρα από τις δυνατότητες εκμάθησης των τυπικών αναδραστικών δικτύων: αλγοριθμικά παραγμένες ακολουθίες που μπορούν να μαθευτούν μόνο από συστήματα με δυνατότητα μνήμης και απαρίθμησης. Για παράδειγμα η σχέση 3.1 μπορεί να παράξει την ακολουθία 3.2, όπου με έντονη γραμματοσειρά σημειώνονται τα στοιχεία της ακολουθίας που μπορούν να προβλεφθούν ντετερμινιστικά.

$$a^n b^n, n > 0 \tag{3.1}$$

$$\text{aab}\mathbf{ba}\text{aab}\mathbf{bba}\text{b}\mathbf{a}\text{aaaab}\mathbf{bbbb} \tag{3.2}$$

Εξετάζονται 4 διαφορετικά μοντέλα: ένα απλό αναδραστικό νευρωνικό δίκτυο, ένα RNN με στοιχεία μνήμης LSTM και 2 RNN με εξωτερική μνήμη. Η εξωτερική μνήμη είναι για το ένα μοντέλο μια διπλά συνδεδεμένη λίστα και για το άλλο μοντέλο μια στοίβα. Όλα τα μοντέλα εκπαιδεύονται με τον αλγόριθμο SGD. Τα μοντέλα με την εξωτερική μνήμη μαθαίνουν να χρησιμοποιούν τις θεωρητικά απείρου μήκους εξωτερικές μνήμες τους, με στοιχειώδεις εντολές (push, pop, insert, no-op).

Τελικώς δείχνουν πως μερικοί βασικοί αλγόριθμοι μπορούν να μαθευτούν από ακολουθιακά δεδομένα χρησιμοποιώντας RNNs με μνήμη. Τα μοντέλα με εξωτερική μνήμη ξεπερνούν σε επιδόσεις τα υπόλοιπα μοντέλα. Οι συγγραφείς υποσημειώνουν πως είναι σημαντικό να μεγαλώσουμε την πολυπλοκότητα του μοντέλου με δομημένο τρόπο και πως η δομή των νευρωνικών δικτύων θα πρέπει να μαθαίνεται από τα δεδομένα και να μην προαποφασίζεται.



Σχήμα 3.1: Παραδείγματα παραγωγής "χειρόγραφων".

3.3 A Synthetic Neural Model for General Purpose Code Generation

Στην έρευνα τους, οι Yin et al. [21] ασχολούνται με την αυτόματη μετατροπή εντολών φυσικής γλώσσας σε πηγαίο κώδικα γλωσσών γενικής χρήσης. Σε αντίθεση με την πλειοψηφία των μεθόδων που απαντώνται στην βιβλιογραφία, που αντιμετωπίζουν το πρόβλημα χωρίς να λαμβάνουν υπ΄ όψιν την γραμματική της τελικής γλώσσας, οι ερευνητές προτείνουν ένα μοντέλο στο οποίο η γραμματική είναι γνωστή a priori.

Το συντακτικά-οδηγούμενο νευρωνικό μοντέλο παραγωγής κώδικα που προτείνεται βασίζεται σε ένα γραμματικό μοντέλο που ορίζει την παραγωγή ενός Abstract Syntax Tree σε ακολουθίες στοιχειωδών δράσεων. Οι δράσεις αυτές χωρίζονται κανόνες παραγωγής κώδικα και σε εντολές. Με αυτό τον τρόπο το μοντέλο δε χρειάζεται να μάθει την γραμματική από τα περιορισμένα σε ποσότητα δεδομένα εκμάθησης. Το αναδραστικό νευρωνικό δίκτυο που χρησιμοποιείται βασίζεται σε στοιχεία LSTM με τροποποίηση, ώστε να λαμβάνεται υπ΄ όψιν η αναδρομική φύση των γλωσσών προγραμματισμού. Η δομή του συστήματος γίνεται σύμφωνα με αρχιτεκτονική encoder-decoder RNN with attention [1], τεχνική η οποία γνωρίζει μεγάλη χρήση και επιτυχία τα λίγα χρόνια ύπαρξης της. Για την εκπαίδευση "δείχνουμε" στο νευρωνικό κομμάτια κώδικα, μετατρέπονται σε ASTs και από εκεί σε κώδικα σύμφωνα την γραμματική που υποδεικνύεται.

Το μοντέλο ξεπερνά τις state of the art προσεγγίσεις νευρωνικών δικτύων στο Hearthstone dataset[4] με παραγόμενη γλώσσα την Python. Συμπεραίνεται έτσι, η σημαντικότητα της γραμματικής της γλώσσας σε σχέση με τις επιδόσεις.

3.4 End-to-End Memory Networks

Οι Sukhbaatar et al., [16] παρουσιάζουν ένα ευέλικτο νευρωνικό μοντέλο με μεγάλη εξωτερική μνήμη. Το μοντέλο σε αντίθεση με αντίστοιχες εργασίες δικτύων με μνήμη εκπαιδεύεται "end-to-end". Αυτό, στα πλαίσια της εκπαίδευσης μοντέλων νευρωνικών δικτύων, σημαίνει πως το μοντέλο εκπαιδεύεται σε μια ενιαία διαδικασία και απλά του δίνονται οι είσοδοι και οι σωστές έξοδοι, χωρίς επιπρόσθετη εργασία για δημιουργία και ρύθμιση χαρακτηριστικών. Οι επιδόσεις του συστήματος εξετάζονται σε προβλήματα συνθετικών ερωταπαντήσεων και σε προβλήματα μοντελοποίησης φυσικής γλώσσας.

Το σύστημα δέχεται ένα σετ εισόδων, μια ερώτηση και εξάγει μία απάντηση. Το σετ εισόδων αποθηκεύεται στη μνήμη σε μορφή εσωτερικών αναπαραστάσεων. Για κάθε ερώτηση υπολογίζεται ένας δείκτης που εκφράζει κατά πόσο αντιστοιχεί η ερώτηση με τα στοιχεία της μνήμης. Από τις εισόδους, επιπρόσθετα, υπολογίζεται και μία αναπαράσταση της αναμενόμενης εξόδου. Η τελευταία σε συνδυασμό με τον δείκτη συσχέτισης ερώτησης-μνήμης χρησιμοποιείται για την εξαγωγή της τελικής απάντησης. Ολόκληρο το σύστημα είναι παραγωγίσιμο, οπότε μπορούμε να χρησιμοποιήσουμε τις τυπικές μεθόδους για την εκμάθηση

[4] `https://github.com/deepmind/card2code`

του.

Για να εξεταστούν τις επιδόσεις στην μοντελοποίηση φυσικής γλώσσας (με την οποία ασχολούμαστε επειδή βρίσκεται πιο κοντά στο πρόβλημα του αυτόματου προγραμματισμού) χρησιμοποιούνται τα Penn Treebank dataset και Text8 dataset. Το δίκτυο μνήμης το οποίο εξετάσαμε ξεπερνά σε επιδόσεις διατάξεις RNN και LSTM. Αξιοσημείωτο είναι πως το νευρωνικό μοντέλο μνήμης έχει σημαντικά λιγότερες παραμέτρους από το αντίστοιχο LSTM. Σε ακόμα ένα πείραμα, έτσι, υποδεικνύεται η σημαντικότητα ύπαρξης εξωτερικής μνήμης στις διατάξεις μάθησης.

3.5 Neuro Symbolic Program Synthesis

Στην έρευνα τους οι Parisotto et al. [11] ασχολούνται με ένα νευρωνικό μοντέλο σύνθεσης προγραμμάτων με σκοπό την επεκτασιμότητα και την εύκολη εξέταση της ορθότητας του παραγώμενου μοντέλου. Σε αντίθεση με την πλειοψηφία των προσεγγίσεων στη σύγχρονη βιβλιογραφία, όπου ο χώρος αναζήτησης είναι σύμβολα της γλώσσας την οποία παράγουμε, εδώ, ο χώρος αναζήτησης είναι υποπρογράμματα που μαθαίνει το σύστημα κατά τη διάρκεια της μάθησης. Το όνομα που δίνεται στο υποσύστημα παραγωγής είναι Recursive-Reverse-Recursive Neural Network (R3NN).

Το υποσύστημα παραγωγής εξάγει αναπαραστάσεις υποπρογραμμάτων σε μορφές δέντρων, στις οποίες κάθε στοιχείο είναι είτε κανόνας παραγωγής είτε σύμβολο, διαδικασία η οποία χωρίζεται σε 3 μέρη. Αρχικά δεδομένου ενός τέτοιου δέντρου, δίνεται ένα διάνυσμα αναπαράστασης σε κάθε φύλλο του. Ύστερα, το δέντρο διαβάζεται προς τα πάνω, ώστε να δοθεί μία αναπαράσταση ολόκληρου του δέντρου στη ρίζα του. Τέλος επαναλαμβάνεται το προς τα κάτω πέρασμα, ώστε να δοθεί σε κάθε φύλλο μια αναπαράσταση ολόκληρου του δέντρου. Με αυτό τον τρόπο, κάθε φύλλο έχει πληροφορία για τα υπόλοιπα φύλλα και για την συνολική λειτουργικότητα του δέντρου. Τα προγράμματα στο σετ δεδομένων χωρίζονται σε στοιχειώδη βήματα για να επεξεργαστούν με τον τρόπο που περιγράψαμε παραπάνω. Τα δεδομένα εκπαίδευσης σε αυτή την περίπτωση αποτελούνται από αναπαραστάσεις εισόδων και εξόδων που δίνονται στο σύστημα παραγωγής με σε κάθε φύλλο του δέντρου.

Το σύστημα εξετάζεται στη δημιουργία προγραμμάτων διαχείρισης αλφαριθμητικών ακολουθιών. Καταφέρνει σε ένα βαθμό και να επεκτείνει προγράμματα που ήδη έχει δει, ώστε να συμπεριλαμβάνουν καινούρια ζευγάρια εισόδου – εξόδου αλλά και να δημιουργήσει καινούρια προγράμματα για καινούριου είδους ζευγάρια εισόδου – εξόδου. Η επεκτασιμότητα που παρουσιάζει το σύστημα αυτό, με την έννοια ότι μπορεί να ξεκινήσει από κάποια προγράμματα και ύστερα να τα εμπλουτίσει, είναι ένα σημαντικό και αισιόδοξο στοιχείο στην κατεύθυνση του αυτόματου προγραμματισμού.

Κεφάλαιο 4

Μεθοδολογία

Στο κεφάλαιο αυτό περιγράφεται η προσέγγιση μας στην παραγωγή κώδικα χρησιμοποιώντας αναδραστικά νευρωνικά δίκτυα. Εμπνεόμαστε από το blog post του Andrej Karpathy[1], στο οποίο χρησιμοποιείται μια σχετικά απλή δομή RNN με LSTM στοιχεία η οποία εκπαιδεύεται στα έργα του Shakespeare, κατά χαρακτήρα, και παράγει παρόμοιο κείμενο. Χρησιμοποιούμε το ίδιο μοντέλο, εκπαιδευμένο σε κώδικα JavaScript. Προτείνουμε μία επέκταση του προηγούμενου μοντέλου που χρησιμοποιεί a priori γνώση για τον κώδικα, με σκοπό να βελτιώσουμε τις επιδόσεις πρόβλεψης του μοντέλου και να εξετάσουμε τη διαίσθηση πως με περισσότερη χρήσιμη πληροφορία ο παραγώμενος κώδικας θα είναι ποιοτικότερος. Εξετάζουμε τα μοντέλα σε 2 διαφορετικά σετ δεδομένων. Παρακάτω ακολουθεί αναλυτική παρουσίαση της μεθόδου, την οποία χωρίζουμε σε 3 στάδια: 1) προ-επεξεργασία, 2) εκπαίδευση και 3) παραγωγή (generation).

4.1 Τα μοντέλα

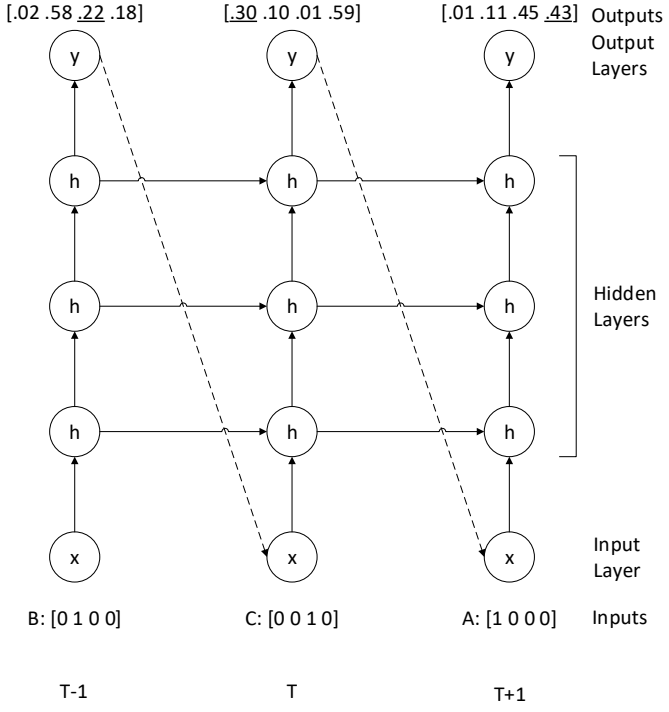
4.1.1 Τα Αναδραστικά Νευρωνικά Δίκτυα ως Μοντέλα Παραγωγής

Ο στόχος της μοντελοποίησης γλώσσας κατά χαρακτήρα (χωρίς να αναφερόμαστε απαραίτητα στην προγραμματιστική γλώσσα) είναι να προβλέψει τον επόμενο χαρακτήρα σε μία ακολουθία. Δεδομένης μιας εκπαιδευτικής ακολουθίας $(x_1, x_2, ..., x_T)$, τα αναδραστικά νευρωνικά δίκτυα χρησιμοποιούν τις εξόδους τους $(o_1, o_2, ..., o_T)$ για να πάρουν κατανομές προβλέψεων της μορφής $P(x_{t+1}|x_{\leq t}) = P(softmax(o_t))$, όπου η κατανομή "softmax" ορίζεται: $P(softmax(o_t) = j) = exp(o_t^{(j)} / \sum_k exp(o_t^{(k)})$. Ο στόχος που χρησιμοποιείται για την μοντελοποίηση της γλώσσας είναι η μεγιστοποίηση της λογαριθμικής πιθανότητας της εκπαιδευτικής ακολουθίας $\sum_{t=0}^{T-1} logP(x_{t+1}|x_{\leq t})$. Όπως και στην εργασία των Graves et al. [5], εισάγουμε στοχαστικότητα δειγματοληπτώντας από την έξοδο του νευρωνικού δικτύου και δίνοντας την τυχαία επιλογή μας ως είσοδο, την επόμενη χρονική στιγμή.

[1] `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`

4.1.2 Μοντέλο char-rnn

Το πρώτο μοντέλο είναι ένα αναδραστικό νευρωνικό δίκτυο με 3 κρυμμένα επίπεδα στοιχείων LSTM. Κάθε στιγμή το σύστημα δέχεται χαρακτήρες κώδικα σε μορφή διανυσμάτων *one-hot* (διανύσματα με όλα τα στοιχεία 0 εκτός από το στοιχείο εκείνο που αντιστοιχεί στον χαρακτήρα και παίρνει την τιμή 1). Ενημερώνει την εσωτερική του κατάσταση και εξάγει μια πρόβλεψη για τον επόμενο χαρακτήρα. Οι προβλέψεις του char-rnn είναι κατανομές του λεξιλογίου, που στην περίπτωση μας αποτελείται από χαρακτήρες. Έστω ότι έχουμε το λεξιλόγιο A, B, C, T. Αν θέλουμε να εκπαιδεύσουμε το σύστημα στην ακολουθία "BCAT", δίνουμε ένα χαρακτήρα τη φορά και θέλουμε να μεγιστοποιηθούν οι υπογραμμισμένες πιθανότητες (Σχήμα 4.1). Στη διαδικασία της παραγωγής (διακεκομμένες γραμμές) δειγματοληπτούμε από τις κατανομές εξόδου για να αποφασίσουμε τον επόμενο χαρακτήρα που δίνεται στο σύστημα.



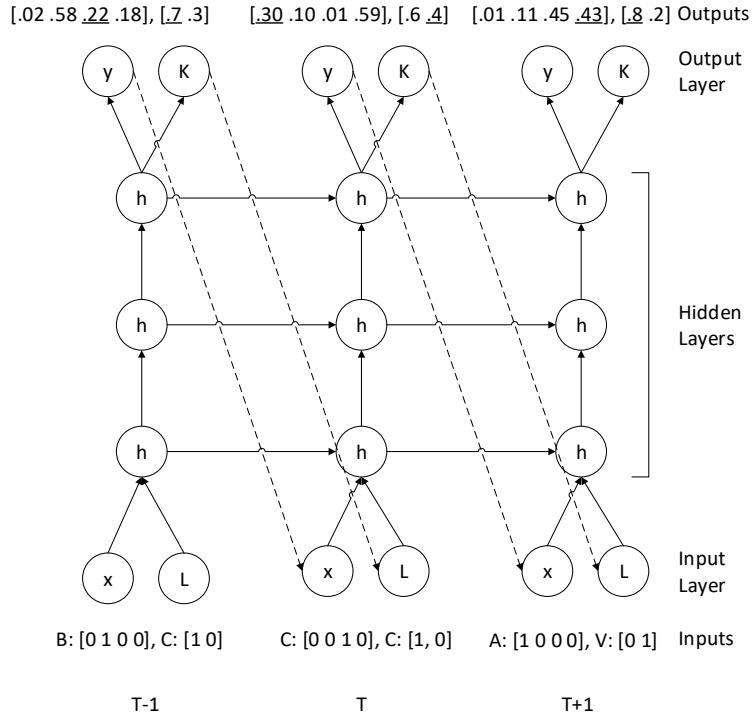
Σχήμα 4.1: Το μοντέλο char-rnn ανεπτυγμένο στο χρόνο.

4.1.3 Μοντέλο labeled-char-rnn

Το δεύτερο μοντέλο είναι επίσης ένα αναδραστικό νευρωνικό δίκτυο με 3 κρυμμένα επίπεδα στοιχείων LSTM. Εκτός από ακολουθίες χαρακτήρων, το μοντέλο αυτό δέχεται και πληροφορία για το είδος του χαρακτήρα. Αντίστοιχα οι έξοδοι του, εκτός από προβλέψεις για τον χαρακτήρα, περιέχουν και προβλέψεις για το είδος του χαρακτήρα. Με τον τρόπο αυτό θα εξετάσουμε κατά πόσο τα RNNs μπορούν να εκμεταλλευτούν a priori γνώσεις για τον κώδικα. Σημειώνεται πως η συνάρτηση επιδόσεων αυτού του μοντέλου είναι γραμμικός

συνδυασμός των επιμέρους επιδόσεων πρόβλεψης χαρακτήρα και είδους χαρακτήρα.

Έστω το λεξιλόγιο A, B, C, T. Έστω επίσης πως δίνουμε το είδος των χαρακτήρων αυτών στο σύστημα με βάση το αν είναι φωνήεντα ή σύμφωνα. Για την εκπαιδευτική ακολουθία "BCAT" δίνουμε την κατάλληλη είσοδο όπως στο σχήμα 4.2. Θέλουμε να μεγιστοποιηθούν και πάλι οι υπογραμμισμένες πιθανότητες. Για την παραγωγή του επόμενου χαρακτήρα και του είδους του, δειγματοληπτούμε από κάθε κατανομή εξόδου ξεχωριστά.



Σχήμα 4.2: Το μοντέλο labeled-char-rnn ανεπτυγμένο στο χρόνο.

4.2 Προ-επεξεργασία

Ο κορμός της διαδικασίας της προ-επεξεργασίας είναι ίδιος και για τα δύο σετ δεδομένων. Αρχικά αναζητούμε τα αρχεία με κατάληξη ".js" διασχίζοντας σειριακά όλους τους φακέλους των projects, εκτός από αυτούς που αφορούν testing και localization. Ο έλεγχος για το τελευταίο γίνεται απλοϊκά, ελέγχουμε δηλαδή αν οι φάκελοι φέρουν τα συνήθη ονόματα που χρησιμοποιούνται για τέτοιου είδους φακέλους. Ο μη ενδελεχής έλεγχος καταφέρνει να αφαιρέσει την πλειοψηφία των επαναλαμβανόμενων αρχείων αφήνοντας ένα μικρό ποσοστό να περάσει. Αυτό έχει ως αποτέλεσμα να εμπλουτιστεί η εκπαίδευση του νευρωνικού, χωρίς όμως να μονοπωλείται το ενδιαφέρον από αρχεία που περιέχουν τετριμμένο κώδικα. Στη συνέχεια, με τη βοήθεια ενός εργαλείου ανάλυσης της σύνταξης και γραμματικής προγραμματιστικών

γλωσσών (ονόματι *linguist*[2]) προχωράμε στο περαιτέρω φιλτράρισμα αρχείων. Συγκεκριμένα, εξαιρούμε αρχεία που έχουν την κατάληξη "*.js*" αλλά δεν είναι αρχεία κειμένου και αρχεία που είναι αυτόματα παραγώμενα και αποτελούν παραπροϊόν της διαδικασίας ανάπτυξης λογισμικού σε JavaScript.

```
1  // is.js
2  // (c) 2001 Douglas Crockford
3  // 2001 June 3
4
5  // is
6  // The -is- object is used to identify the browser.  Every browser edition
7  // identifies itself, but there is no standard way of doing it, and some of
8  // the identification is deceptive. This is because the authors of web
9  // browsers are liars. For example, Microsoft's IE browsers claim to be
10 // Mozilla 4. Netscape 6 claims to be version 5.
11
12 var is = {
13     ie:      navigator.appName == 'Microsoft Internet Explorer',
14     java:    navigator.javaEnabled(),
15     ns:      navigator.appName == 'Netscape',
16     ua:      navigator.userAgent.toLowerCase(),
17     version: parseFloat(navigator.appVersion.substr(21)) ||
18              parseFloat(navigator.appVersion),
19     win:     navigator.platform == 'Win32'
20 }
21 is.mac = is.ua.indexOf('mac') >= 0;
22 if (is.ua.indexOf('opera') >= 0) {
23     is.ie = is.ns = false;
24     is.opera = true;
25 }
26 if (is.ua.indexOf('gecko') >= 0) {
27     is.ie = is.ns = false;
28     is.gecko = true;
29 }
```

Κώδικας 4.1: Αρχείο κώδικα πριν το minification

```
1  var is={ie:navigator.appName=='Microsoft Internet Explorer',java:navigator.
       javaEnabled(),ns:navigator.appName=='Netscape',ua:navigator.userAgent.
       toLowerCase(),version:parseFloat(navigator.appVersion.substr(21))||
       parseFloat(navigator.appVersion),win:navigator.platform=='Win32'}
2  is.mac=is.ua.indexOf('mac')>=0;if(is.ua.indexOf('opera')>=0){is.ie=is.ns=
       false;is.opera=true;}
3  if(is.ua.indexOf('gecko')>=0){is.ie=is.ns=false;is.gecko=true;}
```

Κώδικας 4.2: Αρχείο κώδικα μετά το minification

Αφού επιλέξουμε τα αρχεία τα οποία θα αποτελούν το σετ δεδομένων μας προχωράμε στη διαδικασία της ελαχιστοποίησης του κώδικα (minification, minimisation). Η ελαχιστοποίηση

[2] https://github.com/github/linguist/

Πίνακας 4.1: Παράδειγμα αντιστοιχίας χαρακτήρων με το είδος τους σε μια ακολουθία

String 1	v	a	r		a	=	1	;	f	u	n	c	t	i	o	n		f	(	A	)
Label 1	K	K	K	P	I	O	N	P	K	K	K	K	K	K	K	K	P	I	P	I	P
String 2	{	r	e	t	u	r	n		'	o	k	'	;	}	c	=	f	(	1	0	)
Label 2	P	K	K	K	K	K	K	P	S	S	S	S	P	P	I	O	I	P	N	N	P

κώδικα, είναι η διαδικασία αφαίρεσης περιττών χαρακτήρων από των πηγαίο κώδικα, χωρίς να αλλάζει η λειτουργικότητά του. Τέτοιοι χαρακτήρες είναι τα κενά, τα σύμβολα αλλαγής παραγράφου, τα σχόλια και άλλα. Εδώ χρησιμοποιήθηκε το εργαλείο *jsmin*[3]. Οι κώδικες 4.1, 4.2 δείχνουν ένα αρχείο κώδικα πριν και μετά το minification.

Με την επιλογή αυτή προσπαθούμε να αφαιρέσουμε την περιττή πληροφορία απο τα δεδομένα μας, ώστε να είναι πιο εύκολο για το μοντέλο να αποτυπώσει τις σημαντικές σχέσεις ανάμεσα στους διάφορους χαρακτήρες. Μετά το minification προσθέτουμε 2 ειδικούς χαρακτήρες για την αρχή και το τέλος κάθε αρχείου. Σημειώνεται πως θεωρούμε πως τα αρχεία ειναι extended ASCII κωδικοποιημένα και στην ουσία διαβάζουμε bytes.

Για την εκπαίδευση του μοντέλου labeled-char-rnn χρειάζεται να προετοιμάσουμε με ανάλογο τρόπο την πληροφορία για το είδος των χαρακτήρων. Για το σκοπό αυτό, χρησιμοποιούμε ένα άλλο εργαλείο ανάλυσης σύνταξης και γραμματικής προγραμματιστικών γλωσσών που φέρει το όνομα *pygments*[4]. Η επιλογή για τον διαχωρισμό των ειδών βασίζεται στα αυθαίρετα συνταχτικά δέντρα (abstract syntax trees) της JavaScript, είναι όμως απλουστευμένη και δε χρησιμοποιεί δομές δέντρων, αλλά απλών διανυσμάτων. Ο διαχωρισμός των χαρακτήρων γίνεται ανάμεσα στις ακόλουθες κλάσεις: (**K**eyword, **N**umber, **R**egex, **S**tring, **O**perator, **P**unctuator, **I**dentifier).

Οι χαρακτήρες και τα είδη τους αποθηκεύονται ως λίστες από αλφαριθμητικά στοιχεία ώστε να είναι διαθέσιμα ανά πάσα στιγμή στην εκπαιδευτική διαδικασία. Προφανώς υπάρχει χρονική αντιστοιχία μεταξύ των αρχείων που περιέχουν της ακολουθίες χαρακτήρων με τα αρχεία που περιέχουν το είδος κάθε χαρακτήρα, όπως στα παραδείγματα του πίνακα 4.1. Συνηθίζεται σε τέτοιου είδους προβλήματα να "ανακατεύονται" οι ακολουθίες αλφαριθμητικών χαρακτήρων με σκοπό την . Στα προβλήματα μάθησης έχουμε τη δυνατότητα να "ανακατεύουμε" το σετ δεδομένων με σκοπό την γρηγορότερη/καλύτερη εκπαίδευση των μοντέλων. Επειδή το ζητούμενο μας στη διπλωματική αυτή είναι η παραγωγή κώδικα, και η σειρά των ακολουθιών είναι άρρηκτα συνδεδεμένη με τη λειτουργικότητα και την ουσία των προγραμμάτων, δεν προχωράμε σε αυτή την επιλογή.

4.3 Εκπαίδευση

Η εκπαίδευση γίνεται στο training set του καθενός από τα δύο σετ δεδομένων. Οι χαρακτήρες δίνονται ως *one-hot* διανύσματα, με μήκος όσο και οι διαφορετικοί χαρακτήρες του

[3] `http://www.crockford.com/javascript/jsmin.html`

[4] `http://pygments.org`

σετ δεδομένων. Χρησιμοποιείται η τεχνική του dropout, ενώ ο αλγόριθμος που χρησιμοποιείται για την ελαχιστοποίηση του λάθους είναι ο TBPTT. Η συνάρτηση λάθους είναι η cross-entropy loss function: $\sum_x p(x) \log q(x)$, όπου $p(x)$ είναι η πραγματική κατανομή των χαρακτήρων και $q(x)$ η προβλεπόμενη κατανομή χαρακτήρων του μοντέλου. Η συνάρτηση αυτή χρησιμοποιείται στην πλειοψηφία της σύγχρονης βιβλιογραφίας και εμπειρικά έχει καλά αποτελέσματα στην εκπαίδευση των αναδραστικών νευρωνικών δικτύων. Εξίσου ευρεία χρήση συναντά και η συνάρτηση *rmsprop* που χρησιμοποιούμε για τη βελτιστοποίηση του gradient descent.

Η εκπαίδευση του αναδραστικού νευρωνικού δικτύου γίνεται, πιο περιγραφικά ως εξής: δείχνουμε στο νευρωνικό δίκτυο ακολουθίες σταθερού μήκους, το οποίο προαποφασίζεται της εκπαίδευσης. Ως αληθείς απαντήσεις δίνουμε ένα διάνυσμα ίσου μήκους με το προηγούμενο που περιέχει τους χαρακτήρες της επόμενης χρονικής στιγμής (κύλιση του διανύσματος κατά μία θέση). Στην περίπτωση του μοντέλου labeled-char-rnn με όμοιο τρόπο δίνονται και οι πληροφορίες σχετικά με το είδος των χαρακτήρων, μαζί με τους αντίστοιχους χαρακτήρες. Με σκοπό την παραλληλοποίηση του προγράμματος, δίνουμε πολλά τέτοια παραδείγματα ταυτόχρονα.

Συνολικά εκπαιδεύουμε 4 διαφορετικά μοντέλα. Για κάθε dataset το αντίστοιχο char-rnn και labeled-char-rnn μοντέλο. Για την εκπαίδευση των μοντέλων, πρέπει να αποφασιστεί ένα σύνολο παραμέτρων, που φέρουν σημαντική αξία για τις τελικές επιδόσεις του μοντέλου και την διάρκεια της εκπαίδευσης. Αυτές είναι:

- Μήκος ακολουθίας (Sequence length): Ο αριθμός χαρακτήρων που περιέχει μία ακολουθία.
- Μέγεθος παρτίδας (Batch size): Ο αριθμός των εκπαιδευτικών ακολουθιών που δίνονται παράλληλα στο μοντέλο.
- Μέγεθους κρυμμένων επιπέδων (Hidden state size): Ο αριθμός των στοιχείων LSTM που απαρτίζουν κάθε κρυφό επίπεδο.
- Πιθανότητα dropout: Η πιθανότητα να κρατηθεί ένα στοιχείο στη διάρκεια τη εκπαίδευσης.
- Αριθμός εποχών (Epoch number): Ο αριθμός "περασμάτων" του τεστ δεδομένων.
- Ρυθμός εκμάθησης (Learning rate): Πόσο γρήγορα μαθαίνει το σύστημα από τα λάθη του.

Για την στρατηγική επιλογής και την ακριβή τιμή των υπερ-παραμέτρων θα μιλήσουμε στο Κεφάλαιο 5.

4.4 Παραγωγή

Το μοντέλο που επιλέγουμε για καθένα από τα πειράματα αποφασίζεται σύμφωνα με τις επιδόσεις του στην μετρική λάθους της εκπαίδευσης. Για να είναι ευκολότερα ερμηνεύσιμα

τα αποτελέσματα της εκπαίδευσης, θα χρησιμοποιούμε και την μετρική της "ευστοχίας". Η ευστοχία είναι το ποσοστό επιτυχημένων προβλέψεων επόμενου χαρακτήρα σε μία παρτίδα.

Η διαδικασία παραγωγής κώδικα που περιγράψαμε γενικεύεται και για τα μοντέλα που περιέχουν πληροφορία για το είδος των χαρακτήρων. Μπορούμε δηλαδή να δειγματοληπτούμε από την προβλεπόμενη κατανομή για τα είδη των χαρακτήρων και να χρησιμοποιούμε το αποτέλεσμα ως επόμενη είσοδο. Μπορούμε επίσης να οδηγήσουμε έμμεσα το σύστημα, αρχικοποιώντας το με κώδικα της επιλογής μας. Αυτό αλλάζει την εσωτερική κατάσταση του μοντέλου και το "προϊδεάζει" για το τι κώδικας μπορεί να ακολουθεί. Επιπρόσθετα, κατά τη διάρκεια της δειγματοληψίας έχουμε τη δυνατότητα να επηρεάσουμε την κατανομή που προτείνει το μοντέλο. Αυτό ελέγχει το μοντέλο ως προς τη "σιγουριά" του για τις προβλέψεις του και έχει τη δυνατότητα να κάνει τον παραγώμενο κώδικα, είτε πιο ντετερμινιστικό, είτε πιο ποικίλο. Σημαντική ιδιότητα αυτής της προσθήκης είναι πως δίνει στο μοντέλο τη δυνατότητα να ξεφύγει από φαύλους κύκλους ντετερμινιστικών λαθών χάρη στην επιπλέον τυχαιότητα που εισάγεται. Η συνάρτηση ονομάζεται *Softmax Temperature* και είναι:

$$P = \frac{e^{y/T}}{\sum_{k=1}^{n} e^{y_k/T}} \tag{4.1}$$

Όπου P είναι η νέα κατανομή, y είναι η εξαγόμενη του νευρωνικού δικτύου πιθανοτική κατανομή και n ο αριθμός των διαφορετικών στοιχείων προς πρόβλεψη. T είναι η τιμή της θερμοκρασίας που επηρεάζει την κατανομή. Για τιμές μεγαλύτερες του 1, ο κώδικας γίνεται πιο ποικίλος, αλλά με περισσότερα λάθη. Τιμές μικρότερες του 1 έχουν ως αποτέλεσμα το σύστημα να είναι πιο σίγουρο για τις προβλέψεις του.

Κεφάλαιο 5

Πειράματα και Αποτελέσματα

Στο κεφάλαιο αυτό θα αναλύσουμε τα πειράματα που έγιναν για την εκπαίδευση του μοντέλου παραγωγής κώδικα και θα μελετήσουμε την ποιότητα του παραγώμενου κώδικα. Θα εξετάσουμε ξεχωριστά κάθε σετ δεδομένων και θα συγκρίνουμε τις επιλογές και τις επιδόσεις των δύο προσεγγίσεων σε καθ΄ ένα από αυτά.

5.1 Πειράματα Εκπαίδευσης

Ένα πολύ σημαντικό κομμάτι της εκπαίδευσης ενός τέτοιου συστήματος είναι η κατάλληλη επιλογή των υπερπαραμέτρων. Αποδεικνύεται πως η αποδοτικότερη μέθοδος για την επιλογή τους είναι η τυχαία μέθοδος αναζήτησης [2]. Η υπολογιστική πολυπλοκότητα που εισάγουν τα αναδραστικά νευρωνικά δίκτυα και οι περιορισμένοι υπολογιστικοί πόροι που είχαμε στη διάθεση μας κάνουν την τυχαία μέθοδο εξαιρετικά χρονοβόρα διαδικασία. Αντ΄ αυτού επιλέγουμε εμπειρικά τις υπερ-παραμέτρους (με δοκιμές) και με οδηγό της επιλογές στη σύγχρονη σχετική βιβλιογραφία.

Η αναζήτηση των υπερ-παραμέτρων ξεκίνησε με βάση την πρωτότυπη υλοποίηση του char-rnn. Αυτή χρησιμοποιεί μέγεθος LSTM ίσο με 512 και μήκος ακολουθίας ίσο με 50. Βάσει αυτής αναζητούμε το κατάλληλο μέγεθος παρτίδας, ρυθμό εκμάθησης και την πιθανότητα dropout. Μετά από δοκιμές, τα καλύτερα αποτελέσματα στο σετ επιβεβαίωσης προκύπτουν για μέγεθος LSTM 200, ρυθμό εκμάθησης 0.02 και 20% πιθανότητα dropout. Τα τελικά μοντέλα επιλέγουμε να είναι μεγαλύτερα σε μέγεθος, επειδή καθ΄ όλη τη διάρκεια των δοκιμών παρουσιάζεται σημαντικό underfitting. Η διακοπή της εκπαίδευσης γίνεται με τη μέθοδο *early stopping*. Σημειώνεται πως οι επιλεγμένες παράμετροι μπορεί να ευνοούν το μοντέλο char-rnn λόγω των παραπάνω αποφάσεων. Η εξαντλητική αναζήτηση των υπερ-παραμέτρων με τη μέθοδο τυχαίας αναζήτησης όμως, θα διαρκούσε τουλάχιστον λίγους μήνες οπότε και δεν τη χρησιμοποιούμε.

5.1.1 100 Δημοφιλέστερα Github JavaScript Projects Πειράματα

Το σετ δεδομένων αυτό αποτελείται από τα 100 πιο δημοφιλή projects σε γλώσσα JavaScript στον ιστότοπο αποθετηρίων λογισμικού github[1]. Μετά το preprocessing παίρνουμε ακολουθίες συνολικού μήκους περίπου 79 εκατομμυρίων χαρακτήρων. Υπάρχουν 212 διαφορετικοί χαρακτήρες, συμπεριλαμβανομένων των ειδικών χαρακτήρων αρχής και τέλους αρχείων. Χρησιμοποιούμε το 95% των δεδομένων για την εκπαίδευση του συστήματος και το υπόλοιπο 5% για την επικύρωση της μάθησης. Λόγω της έλλειψης τεστ σετ δεν εξασφαλίζεται πως τα αποτελέσματα μας δεν κάνουν overfit στα δεδομένα επικύρωσης. Αυτό όμως είναι δευτερευούσης σημασίας αφού στόχος μας είναι να παράξουμε κώδικα και δεν υπάρχουν διαθέσιμα συγκριτικά αποτελέσματα (benchmark results) για τον σκοπό αυτό.

Η στρατηγική επιλογής των παραμέτρων έχει ως εξής: Για να είναι οι δύο προσεγγίσεις συγκρίσιμες κρατάμε ίδιο το μέγεθος των κρυφών επιπέδων. Από αυτή την επιλογή εξαρτάται κυρίως ο αριθμός συνολικών παραμέτρων προς εκπαίδευση. Για το πρώτο σετ δεδομένων αποφασίζουμε τον αριθμό αυτό σε 1024, αριθμός αρκετά μεγάλος ώστε να είναι αντιμετωπίσιμο από το σύστημα το ογκώδες σετ δεδομένων.

Η επόμενη υπερ-παράμετρος που πρέπει να αποφασιστεί είναι το μήκος της εκπαιδευτικής ακολουθίας, η μεταβλητή k_2 του αλγορίθμου TBPTT. Η υπερ-παράμετρος αυτή έχει μεγάλη σχέση τόσο με την ποιότητα του παραγώμενου κώδικα, αφού ελέγχει πόσους από τους προηγούμενους χαρακτήρες "βλέπει" το σύστημα, αλλά και με τον χρόνο εκτέλεσης μιας εποχής, αφού μεγαλύτερες ακολουθίες εισάγουν υπολογιστική πολυπλοκότητα. Το μέγεθος των εκπαιδευτικών ακολουθιών αποφασίζεται στους 100 χαρακτήρες και για τα δύο μοντέλα.

Ο ρυθμός εκμάθησης είναι άμεσα συνδεδεμένος με το μέγεθος παρτίδας. Όσο περισσότερα παραδείγματα βλέπει ταυτόχρονα το σύστημα τόσο πιο σίγουρο θα πρέπει να είναι για τα συμπεράσματα του. Εξαγωγή δυνατών συμπερασμάτων από λιγοστά παραδείγματα πρέπει να αποφεύγεται. Τελικώς δείχνουμε 200 ακολουθίες σε κάθε βήμα εκμάθησης και θέτουμε τον ρυθμό εκμάθησης στην τιμή 0.002, ώστε να γεμίζουμε όσο καλύτερα γίνεται την μνήμη του υπολογιστικού συστήματος αλλά ταυτόχρονα να συνεχίσουμε να μαθαίνουμε αποτελεσματικά. Σημειώνεται πως η προτεινόμενη τιμή για τον ρυθμό εκμάθησης της rmsprop είναι το 0.001.

Τέλος, επειδή το σετ δεδομένων είναι αρκετά ογκώδες και περίπλοκο, είναι δύσκολο το μοντέλο μας να κάνει overfit. Έτσι, δε χρειάζεται η πιθανότητα dropout να είναι εξαιρετικά μεγάλη. Επιλέγουμε την υπερ-παράμετρο αυτή στο 20%, ενώ η γενική προτεινόμενη τιμή είναι 40% με 50%. Ο αριθμός των εποχών αποφασίζεται έτσι ώστε κανένα από τα 2 μοντέλα να μη βελτιώνει τις επιδόσεις του στο σετ δεδομένων επιβεβαίωσης. Ο αριθμός αυτός προκύπτει στις 60 εποχές. Στον πίνακα 5.1 παρουσιάζονται συνοπτικά οι παραπάνω αποφάσεις.

Η εκπαίδευση έγινε σε μία κάρτα γραφικών Nvidia Gtx 960 με 4 gb RAM με τη βοήθεια της βιβλιοθήκης keras[2]. Η εκπαίδευση διαρκεί 6 περίπου ημέρες για το πρώτο μοντέλο και 7 περίπου για το δεύτερο. Όπως αναφέραμε, η παρακολούθηση των επιδόσεων και η επιλογή των σετ βαρών για την παραγωγή κώδικα γίνεται σύμφωνα με την μετρική average cross entropy

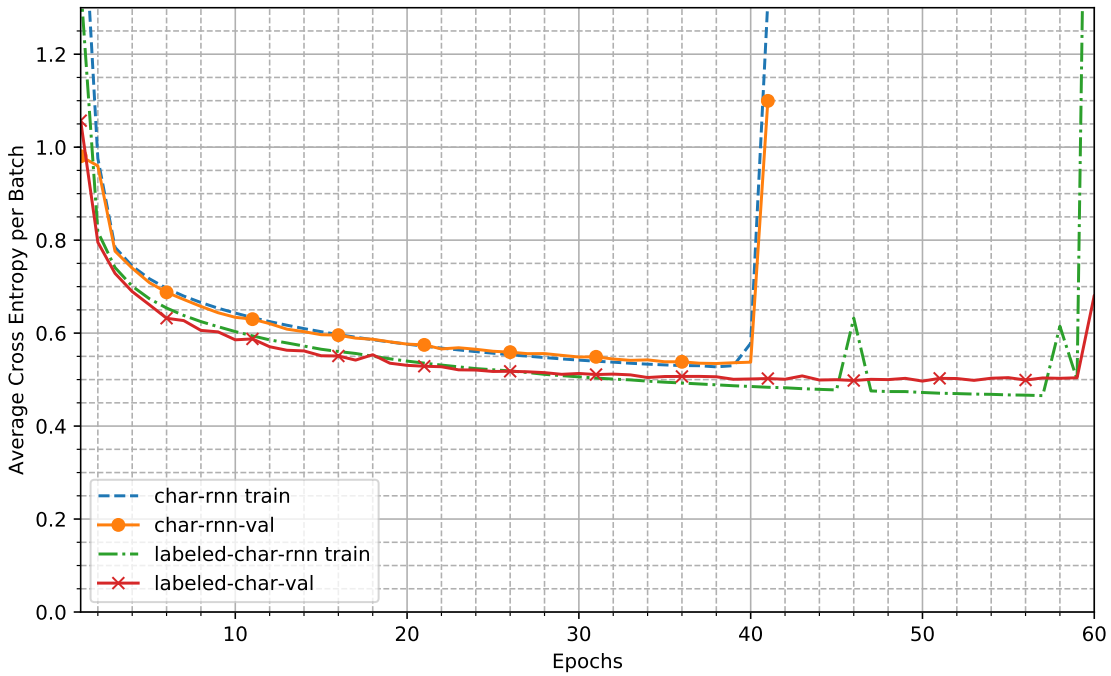
[1] `www.github.com`

[2] `https://keras.io`

Πίνακας 5.1: Υπερπαράμετοι για τα top 100 Github js projects

	char-rnn	labeled-char-rnn
# Παραμέτρων	23M	23M
# Χαρακτήρων	212	212, 8
# Εποχών	40	60
Μέγεθος LSTM	1024	1024
Μήκος Ακολουθίας	100	100
Ρυθμός Εκμάθησης	0.002	0.002
% Dropout	20	20
Μέγεθος Παρτίδας	200	200

per minibatch. Σημειώνεται πως η σύγκριση των μοντέλων στην μετρική αυτή γίνεται μόνο στο κομμάτι που αφορά την πρόβλεψη χαρακτήρων. Στην εικόνα 5.1 φαίνεται η εξέλιξη της εκπαίδευσης των 2 μοντέλων στην περίοδο 40 και 60 εποχών στο σετ εκπαίδευσης και το σετ επαλήθευσης. Ως μοντέλα παραγωγής, επιλέγουμε αυτά με τα βάρη τις 38ης εποχής για το μοντέλο char-rnn και της 53ης εποχής για το μοντέλο labeled-char-rnn. Οι επιδόσεις τους συνοψίζονται στον πίνακα 5.2. Τα αποτελέσματα της εκπαιδευτικής διαδικασίας είναι σε πρώτη όψη ικανοποιητικά. Οι καμπύλες εκπαίδευσης και επαλήθευσης μένουν σε κοντινά επίπεδα και για τα δύο μοντέλα, γεγονός που μαρτυρά καλή γενίκευση των χαρακτηριστικών που μαθαίνονται. Η ευστοχία, ιδιαίτερα, της ανάθεσης είδους στον επόμενο χαρακτήρα κυμαίνεται σε πολύ υψηλά επίπεδα.



Σχήμα 5.1: Καμπύλες εκμάθησης για τα 100 δημοφιλέστερα github js projects

Πίνακας 5.2: Επιλεγμένα μοντέλα

	Εποχή	Λάθος Επιβεβαίωσης	% Ευστοχίας
char-rnn	38	0.534	85.6
labeled-char-rnn	53	0.526, 0.11	87.2, 97.1

Πίνακας 5.3: Υπερπαράμετοι για τα top 200 npm js projects

	char-rnn	labeled-char-rnn
# Παραμέτρων	10M	10M
# Χαρακτήρων	210	210, 8
# Εποχών	80	80
Μέγεθος LSTM	512	512
Μήκος Ακολουθίας	100	100
Ρυθμός Εκμάθησης	0.002	0.002
% Dropout	30	40
Μέγεθος Παρτίδας	200	200

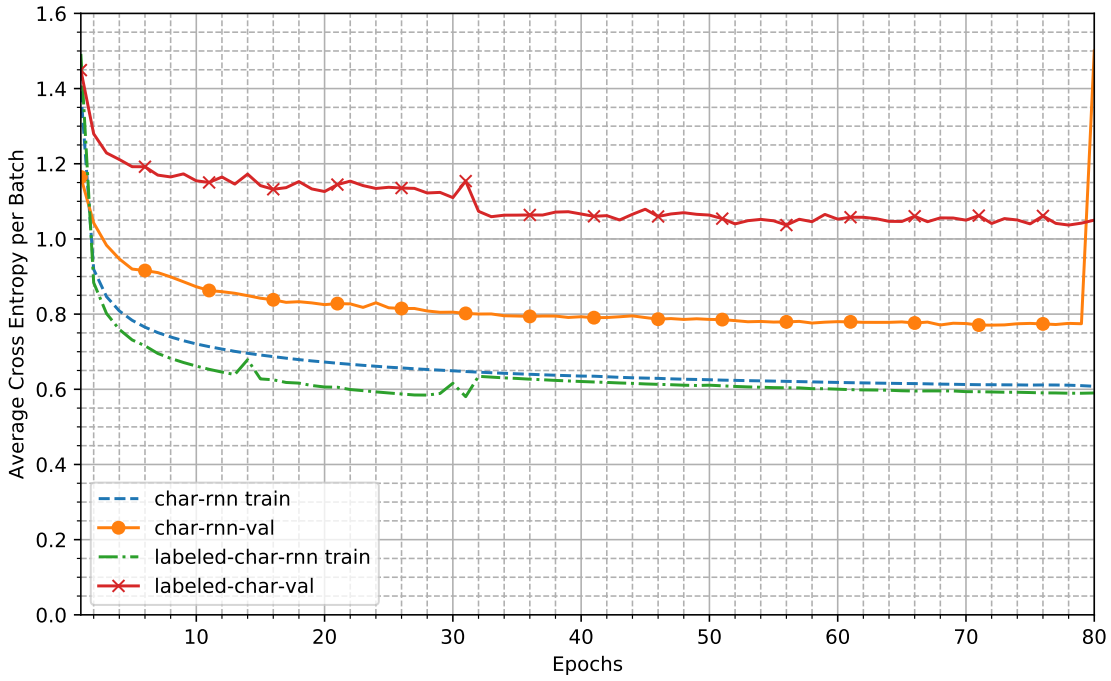
5.1.2 200 Δημοφιλέστερα npm Projects Πειράματα

Το δεύτερο σετ δεδομένων αποτελείται από τις 200 πιο δημοφιλείς βιβλιοθήκες javascript του ιστοχώρου www.npmjs.com. Οι ακολουθίες μετά την προ-επεξεργασία αριθμούν περίπου 49 εκατομμύρια χαρακτήρες με 210 διαφορετικούς χαρακτήρες. Σε αυτό το πείραμα χωρίζουμε το 90% των ακολουθιών στο σετ εκπαίδευσης και το 10% στο σετ επαλήθευσης, επειδή έχουμε λιγότερα δεδομένα και θέλουμε να αποφύγουμε μεγάλη διακύμανση στο σετ επαλήθευσης.

Οι αποφάσεις των υπερ-παραμέτρων βασίζονται στις παρατηρήσεις μας από τα προηγούμενα πειράματα. Έτσι, κρατάμε ίδιο το μέγεθος παρτίδας, τον ρυθμό εκμάθησης και το μήκος ακολουθίας. Το σετ εκπαίδευσης έχει μικρότερο μέγεθος από το προηγούμενο πείραμα και από τις πρώτες δοκιμές παρατηρούμε σημαντικό overfitting. Προς την κατεύθυνση καλύτερης γενίκευσης των συμπερασμάτων του συστήματος, αρχικά μικραίνουμε το δίκτυο θέτοντας το μέγεθος LSTM σε 512, κίνηση η οποία μειώνει σημαντικά τις εκπαιδεύσιμες παραμέτρους του συστήματος. Έπειτα αυξάνουμε την πιθανότητα dropout σε 30% και 40% που αποτέλεσμα έχει την αργότερη εκπαίδευση του αναδραστικού νευρωνικού δικτύου. Για να αποζημιώσουμε την τελευταία μας επιλογή αυξάνουμε τις εκπαιδευτικές εποχές των μοντέλων στις 80. Οι εκπαιδευτικές επιλογές συνοψίζονται στον πίνακα 5.3.

Η διαδικασία που ακολουθείται είναι η ίδια με του προηγούμενου πειράματος, δηλαδή εκπαιδεύουμε το σύστημα σε μία κάρτα γραφικών Nvidia Gtx 960 με 4 gb RAM και επιλέγουμε το μοντέλο με τις καλύτερες επιδόσεις στη μετρική πρόβλεψης χαρακτήρων. Η εκπαίδευση του char-rnn διαρκεί 3 ημέρες ενώ του labeled-char-rnn διαρκεί περίπου 4. Η εξέλιξη της εκπαίδευσης φαίνεται στην εικόνα 5.2.

Το επιλεγόμενο μοντέλο για το char-rnn είναι αυτό της 72ης εποχής και για το labeled-char-rnn το επιλεγόμενο μοντέλο είναι αυτό της 78ης εποχής. Τα αποτελέσματα συνοψίζονται



Σχήμα 5.2: Καμπύλες εκμάθησης για τα 200 δημοφιλέστερα npm js projects

Πίνακας 5.4: Επιλεγμένα μοντέλα

	Εποχή	Λάθος Επιβεβαίωσης	% Ευστοχίας
char-rnn	72	0.771	78.9
labeled-char-rnn	78	1.038, 0.161	72.3, 94.7

στον πίνακα 5.4. Είναι εμφανές από το διάγραμμα ότι τα μοντέλα μας δυσκολεύονται περισσότερο να γενικεύσουν τα συμπεράσματα που εξάγουν από αυτό το σετ δεδομένων. Η ικανότητα πρόβλεψης του είδους του επόμενου χαρακτήρα παραμένει σε σχετικά υψηλά επίπεδα αλλά η προσθήκη της δεν βελτιώνει τις επιδόσεις στο σετ επιβεβαίωσης. Θα εξετάσουμε αναλυτικότερα τα αποτελέσματα αυτά στο υποκεφάλαιο των αποτελεσμάτων και στο κεφάλαιο των συμπερασμάτων.

5.2 Αποτελέσματα

Χρησιμοποιούμε τα μοντέλα που εκπαιδεύτηκαν παραπάνω για να παράξουμε 100 αρχεία κώδικα για κάθε προσέγγιση και κάθε ομάδα μοντέλων. Επιλέγουμε ένα JavaScript project με το οποίο αρχικοποιούμε κάθε ομάδα μοντέλων. Η αρχικοποίηση του μοντέλου γίνεται με σκοπό την έμμεση οδήγηση του. Αρχικά θα εξετάσουμε την ποιότητα του κώδικα εποπτικά και έπειτα θα χρησιμοποιήσουμε το εργαλείο *jshint*[3] για να κάνουμε στατική ανάλυση του κώδικα.

[3] `http://jshint.com`

5.2.1 100 Δημοφιλέστερα Github Javascript Projects Παραγώμενος Κώδικας

Επιλέγουμε ένα project που δεν έχει εμφανιστεί στη διάρκεια της εκπαίδευσης και της επαλήθευσης. Συγκεκριμένα επιλέγουμε το hyper terminal που είναι το πιο δημοφιλές JavaScript project στο github το πρώτο εξάμηνο του 2017. Ο κώδικας 5.1 είναι ένα αρχείο από το project αυτό.

```
1  const valsCache = new WeakMap();
2
3  export function values(imm) {
4    if (!valsCache.has(imm)) {
5      valsCache.set(imm, Object.values(imm));
6    }
7    return valsCache.get(imm);
8  }
9
10 const keysCache = new WeakMap();
11 export function keys(imm) {
12   if (!keysCache.has(imm)) {
13     keysCache.set(imm, Object.keys(imm));
14   }
15   return keysCache.get(imm);
16 }
```

Κώδικας 5.1: Δείγμα κώδικα απο το hyper terminal

Οι κώδικες 5.2, 5.3 είναι δημιουργήματα των μοντέλων char-rnn και labeled-char-rnn αντίστοιχα. Τα αρχεία αυτά επιλέχτηκαν χάρη στο μικρό μέγεθός τους και την συντακτική ορθότητα.

Ήδη εποπτικά παρατηρούμε τη δυνατότητα και των δύο μοντέλων να αναπαράγουν συντακτικές δομές, όπως οι παρενθέσεις και οι αγκύλες, αλλά και λογικές, όπως οι συναρτήσεις και οι δομές πολλαπλών επιλογών. Φαινομενικά αυτό είναι ένα συνηθισμένο αρχείο javascript. Με μία δεύτερη, αναλυτικότερη ματιά παρατηρούμε σημαντικά λάθη στη χρήση μη ορισμένων μεταβλητών (5.2, γραμμή 3), την ύπαρξη γραμματικών λαθών (5.2, γραμμή 23) και ενέργειες χωρίς αποτέλεσμα (5.2, γραμμή 21). Είναι εύκολο να συμπεράνει κανείς, ίσως και χωρίς να είναι γνώστης της γλώσσας, πως τα προγράμματα αυτά δεν θα καταφέρουν να μεταφραστούν.

```
"aabbx";
WebInspector.ElementsTreeElement.parseChildren = function (
    fileNameBeforeReplace) {
    this.name = name;
    this.name = parseName(name, "x");
    if (!this._absoluteRegex)
        this._htmlEditsExpanded = true;
}
WebInspector.CodeMirrorResults.prototype = {
    reset: function () {
        this._addPlainHeaders(this);
    },
    componentLayout: function (registerElement) {
        this._logErrorMessage = window.InvalidError;
        this._showCompiler.registerElement(getFragment, registerElement);
        this._title = registerElement;
        this._killBehaviorsInNavigating = true;
        this._createView();
    },
    _enableCandidate: function (event) {
        if (!WebInspector.extensionAPI.ResourceContainer.excludeOnDominated
            ())
            return;
        this._auditSearchResult.viewFactory = eventData;
        this._executionContextsDefaultAction = active + (event.data.name >=
             WebInspector.isSettingsUI.isControlled() ? event.section :
            WebInspector.UIString("Back"));
    },
    __proto__: WebInspector.VBox.prototype
}
```

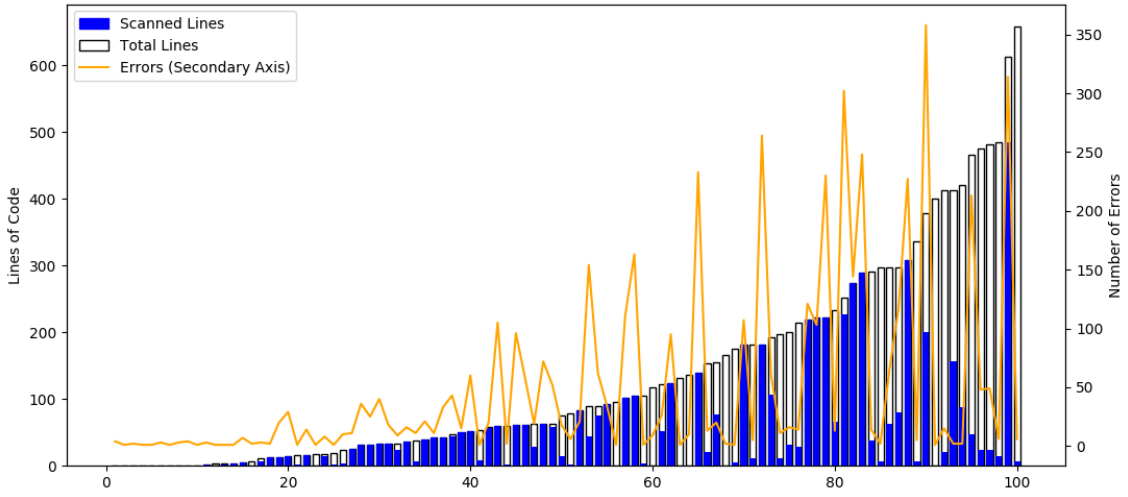
Κώδικας 5.2: Δείγμα κώδικα από το char-rnn

Για την καλύτερη εκτίμηση των αποτελεσμάτων των νευρωνικών δικτύων, και για τη σύγκριση των δύο θα ακολουθήσουμε μια πιο ποσοτική προσέγγιση. Με τη χρήση του εργαλείου ανάλυσης κώδικα *jshint* θα εξετάσουμε τον αριθμό των συντακτικών και διαφόρων άλλων λαθών (Errors) που εντοπίζονται και μέχρι πιο σημείο καταφέρνουν να αναγνωστούν πριν βρεθεί ένα ανεπανόρθωτο συντακτικό λάθος (% Scanned Lines). Σημειώνεται πως τα μοντέλα αποφασίζουν αυτόνομα το μήκος του κώδικα, με τη χρήση των ειδικών χαρακτήρων, αλλά τίθεται ένα άνω όριο 15000 χαρακτήρων στο οποίο θεωρούμε ότι το αρχείο ξεφεύγει διαχειρισιμότητας λόγω μεγέθους. Στις εικόνες 5.3 - 5.8 φαίνονται τα αποτελέσματα της παραπάνω ανάλυσης και το μήκος των παραγώμενων αρχείων.

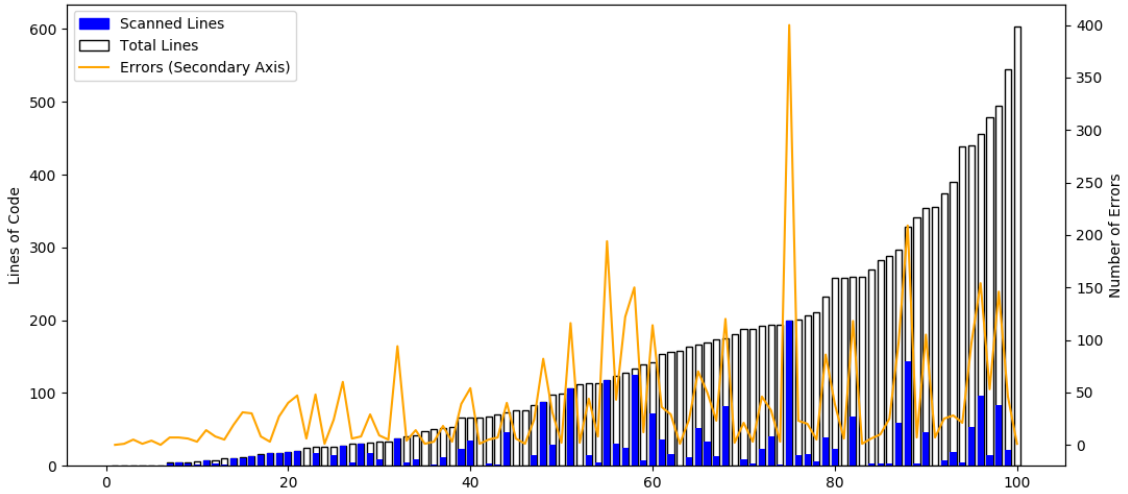
```
function HardwareTagThrowgTear(existingTarget) {
    this.target = ense targetFactory(bs);
}
version.THREE.RegistryBackground = ThrottleEscane = TTRIGNORES;
versionmTurbilares = frames, self = truncate(+c * s, vAs, helpers.body,
    head || self.Meteor);
}
get relativeTo(
    if (relativeAtRuntime)
        console.log("\n\n<%= setTimeout(blockVersion? %2; %> %s (%s).",
            newTime, timeInfo.done); preload(fallback,
        continue);
}, fullNameToUsername: function (rule) {
    var pref = pref && rule.prefix && rule.rule;
    switch (self.timeout) {
    case "test":
        return self._abs ? test.isTest(rule);
        besoCreate();
        break;
    case "fake":
        info.done = true;
        break;
    case "test":
        if (resource.getType() == "file") {
            info.getPrefFeature = word);
        return name;
    }
    for (version = 5; i < print; version++) {
        var match = result[fn.toUpperCase()];
        inferenceText(message, i);
        id == 0 ? "[" + (i - p > M.$op) ? genWindowsNDFD(test) : ""];
}
vateStyleSheetForText(rSelectorPaneMarkers);
if (docElem.styleSheets) {
    This.styleRuleF(0, msg, expectingHashFnParts(thandleMethodCaller));
}
expandReady(cssRules, expandCache, [fileExpandType())];
}, setDocumentWb2Value: findRule(const styleSheet, files), styleSheets.
    length && styleSheets.push(firstContent)
};
};
findMediaStyles();
href.tabPanes = className;
};
});
```

Κώδικας 5.3: Δείγμα κώδικα απο το labeled-char-rnn

Στα σχήματα 5.3 και 5.4 μπορούμε να συγκρίνουμε τις επιδόσεις των δύο μοντέλων. Τα στατιστικά των 100 αρχείων έχουν ταξινομηθεί κατά τον αύξοντα αριθμό γραμμών. Αρχικά παρατηρούμε πως και τα δύο μοντέλα παράγουν κάποια κενά αρχεία ή αρχεία με 1 ή 2 γραμμές.



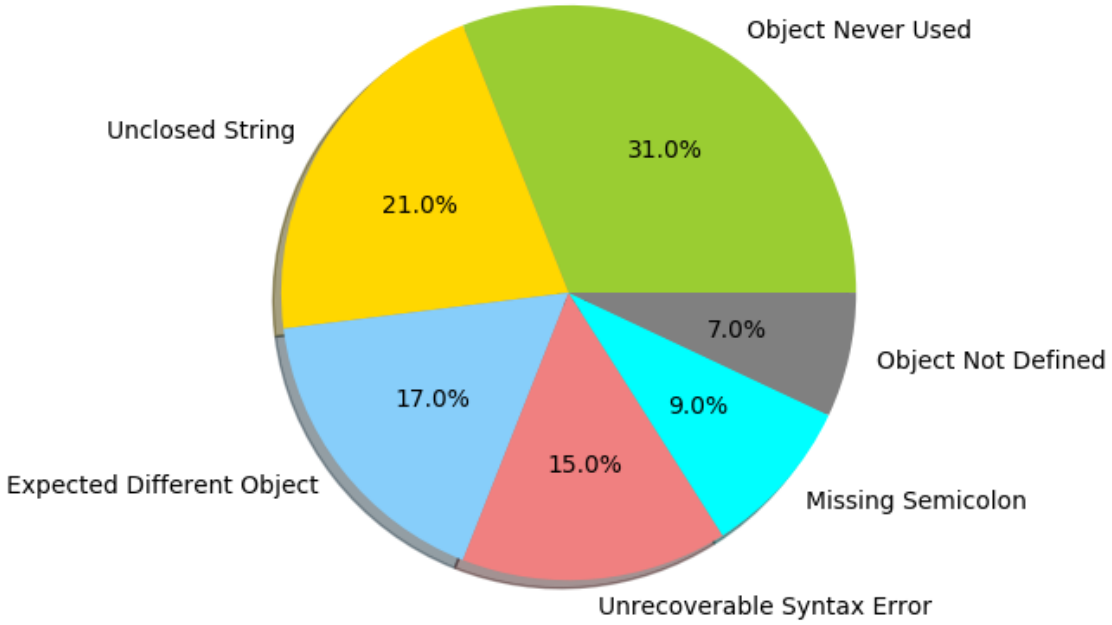
Σχήμα 5.3: Στατική ανάλυση κώδικα για τα αποτελέσματα του char-rnn μοντέλου



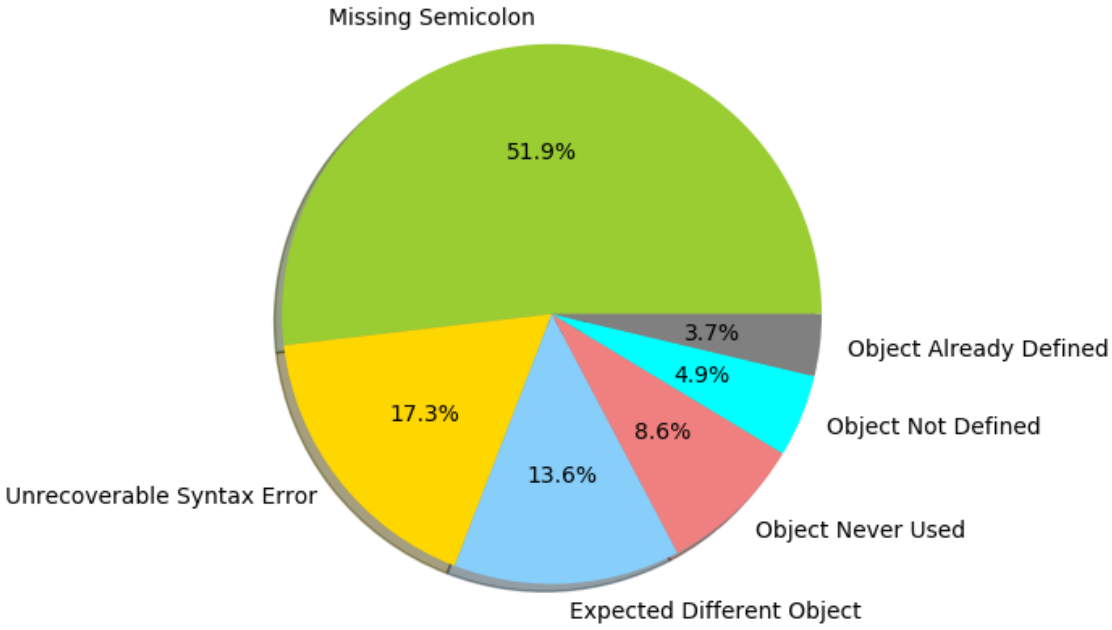
Σχήμα 5.4: Στατική ανάλυση κώδικα για τα αποτελέσματα του labeled-char-rnn μοντέλου

Το μέσο μήκος των αρχείων είναι αρκετά κοντινό (137 γραμμές char-rnn – 142 γραμμές labeled-char-rnn) και το δεύτερο μοντέλο καταφέρνει να κλείσει τα αρχεία σε λιγότερο από 15000 χαρακτήρες 9% των περιπτώσεων ενώ το πρώτο 6%. Αν και το labeled-char-rnn έχει περισσότερη πληροφορία για τον κώδικα, παράγει αρχεία που ολοκληρώνουν την ανάλυση κώδικα με αρκετά μικρότερη συχνότητα. Η γενική εικόνα που παίρνουμε από την ανάλυση αυτή είναι απογοητευτική για το δεύτερο μοντέλο, που ενώ έχει a priori γνώση για τη γλώσσα αποτυγχάνει να την αποτυπώσει με τρόπο που να παράγει ποιοτικά αποτελέσματα.

Τα είδη των συνηθέστερων λαθών μαρτυρούν τις αδυναμίες της προσέγγισης μας στην παραγωγή κώδικα. Ακόμα και αν καταφέρνουν να δημιουργηθούν μικρά κομμάτια λειτουργικού κώδικα η έλλειψη ανεξάρτητης μνήμης και η στοχαστική φύση της παραγωγής δυσκολεύουν τα μοντέλα ως προς τη διαχείριση των διαφόρων μεταβλητών, τη χρήση της σύνταξης και τη επίτευξη του τελικού σκοπού. Αυτή είναι η κύρια αιτία των λαθών Object Already Defined,

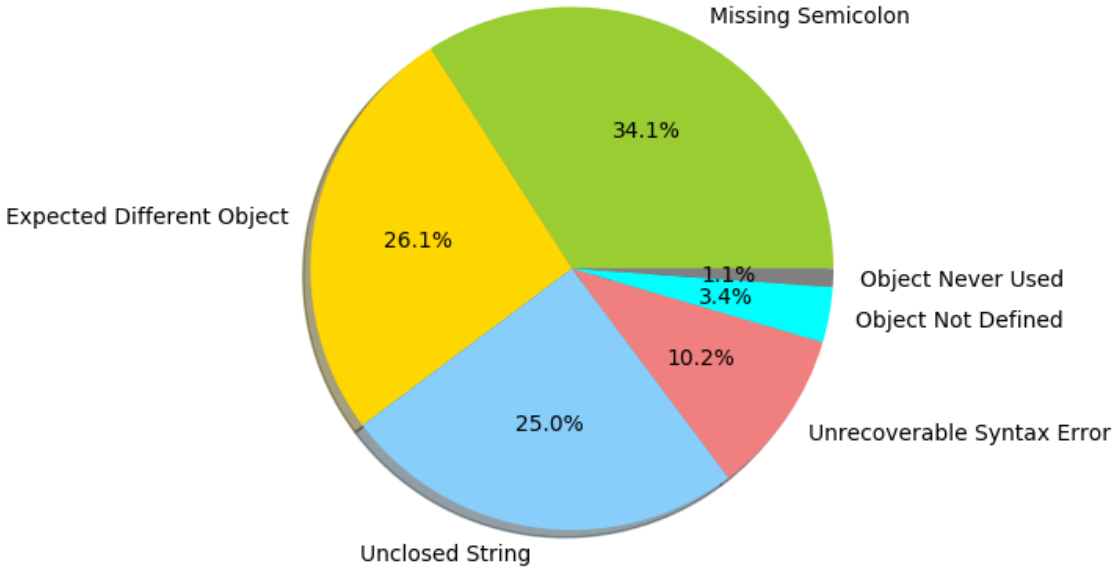


Σχήμα 5.5: Συνηθέστερο λάθος των αρχείων του char-rnn

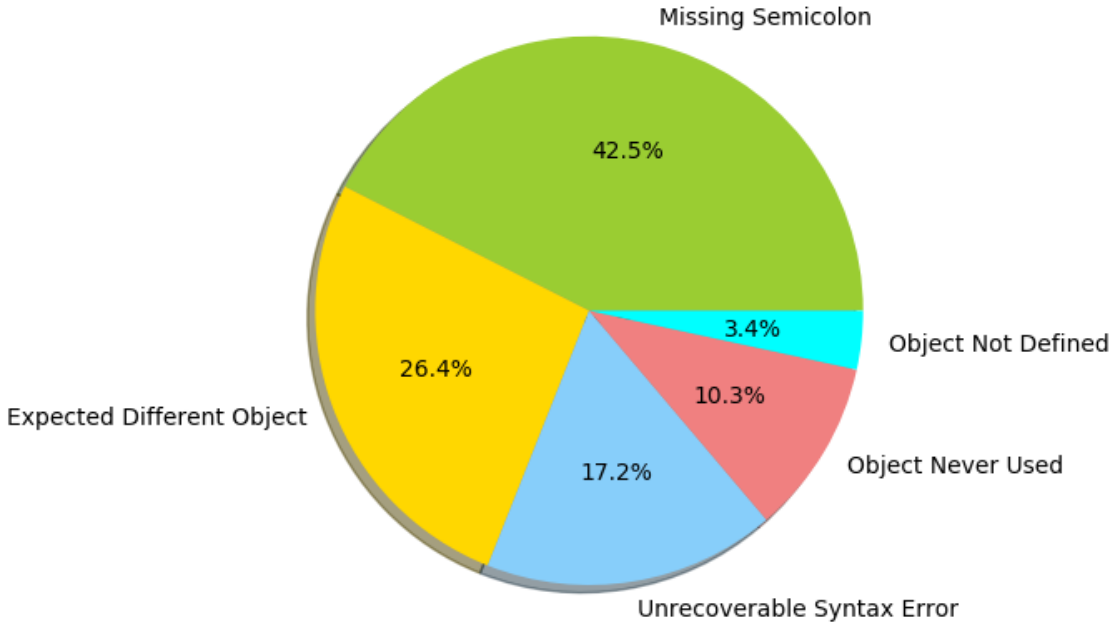


Σχήμα 5.6: Δεύτερο συνηθέστερο λάθος των αρχείων του char-rnn

Object Not Defined, Object Never Used, Expected Different Object. Αρκετά συχνά η δημιουργία αλφαριθμητικών ξεφεύγει της διαχείρισης του μοντέλου και συνεχίζεται να γράφεται κώδικας μέσα σε κάποιο αλφαριθμητικό - που είναι ο κύριος λόγος της ύπαρξης του Unclosed String σε συνδυασμό με το Missing Semicolon.



Σχήμα 5.7: Συνηθέστερο λάθος των αρχείων του labeled-char-rnn



Σχήμα 5.8: Δεύτερο συνηθέστερο λάθος των αρχείων του labeled-char-rnn

5.2.2 200 Δημοφιλέστερα npm Projects Παραγώμενος Κώδικας

Ακολουθούμε ανάλυση όμοια με την προηγούμενη υποενότητα. Το project αρχικοποίησης που χρησιμοποιείται είναι το lodash, μια βιβλιοθήκη της JavaScript που διευκολύνει τη διαχείριση διανυσμάτων, αντικειμένων, αλφαριθμητικών και τη διαδικασία του unit testing. Ο κώδικας 5.4 είναι του ιδίου project και οι κώδικες 5.5, 5.6 είναι επιλεγμένοι κώδικες των παραγώμενων μοντέλων.

```
function filter(array, predicate) {
  let index = -1
  let resIndex = 0
  const length = array == null ? 0 : array.length
  const result = []

  while (++index < length) {
    const value = array[index]
    if (predicate(value, index, array)) {
      result[resIndex++] = value
    }
  }
  return result
}

export default filter
```

Κώδικας 5.4: Δείγμα κώδικα απο το lodash

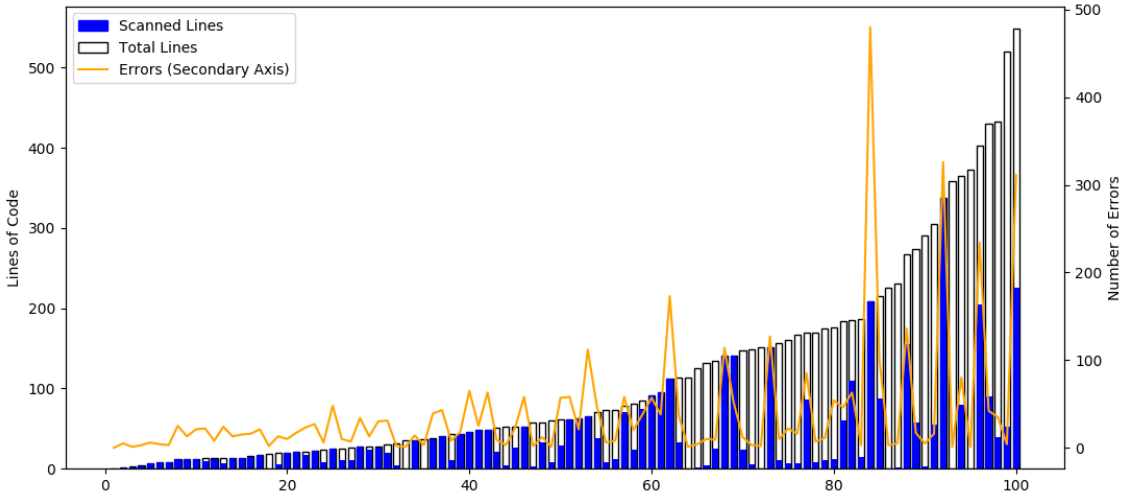
```
WebInspector.ColumnIndexedChild = function (nodeId) {
    this._rootCollection = new InspectorFrontendHost(this._useLatestURL.
        apply(this._addRootNode), root);
    if (WebInspector.CSSHrafNode.removeShadowRootPayload(node))
        WebInspector.initItemCreateProperty(addRangeChildRecord.name,
            createElement("[disabled.added"]));
if (resource) {
    this._baseShortcut = domModel;
    dispatchConfig.postingRunning = nodeURL;
}
splitOnClick = resourceLoader.platform;

function baseRestAppended(shortcut, initiator) {
    function callFrameParts(callback, shortcut, scale) {
        return typeof callback === "function" && callback;
    }

    function newCallFrames(callback) {
        try {
            var node = node.getMatchedSize;
            return callback.bind(this, savedArgs, subCollection, ownerStyle
                );
        }
    };
```

Κώδικας 5.5: Δείγμα κώδικα απο το char-rnn

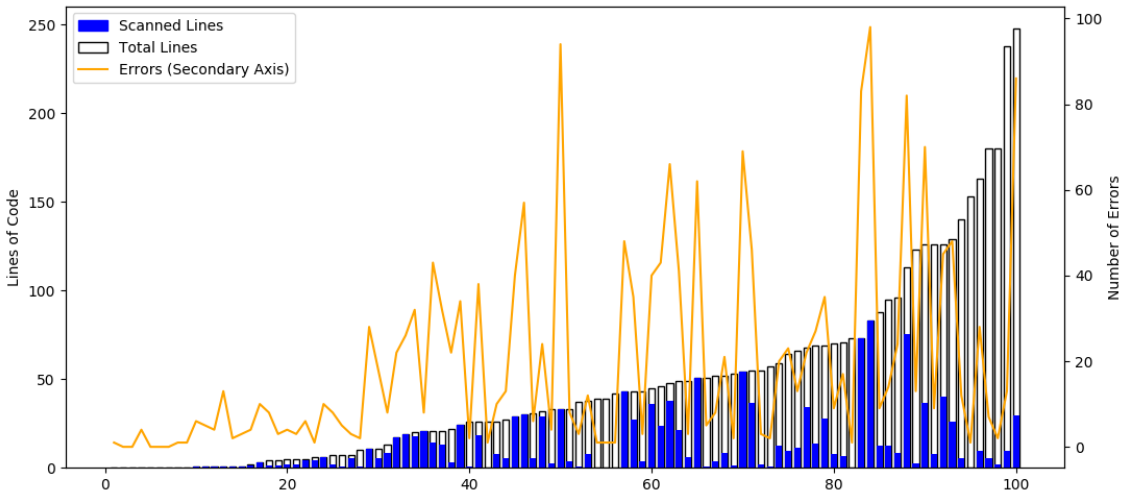
```
vers(0, join) {
    if (!_.isString(input)) {
        let require = 2. input;
        toString = test;
        relse
        if (index === undefined && !isMaske);
        r
        return Joi.config(input, inner).convert('asnum.json');
    };
    test('validate:quoteCheck', _fhend, void 0);
}

function refixedUniq(ncompress, asahOp) {
    if (arguments.length > 0) {
        a = Promise;
    }
    return aspect;
}
fireadsoafiest = first.collection;
var asove = Series;
validate.call();

function Pool() {
    return {
        anonymousIdeautountMOpen: ashify(anonymousOptions),
        _agunin: as;
    };
    KeyboardEvent.prototype.bind = f;
    valsea(ASIPLf);
    Stotin(AsyNode, applyOptions);
    inherits(Domain, Object.behaviors);
    Association.Promise.TimeoutType = Domain;
```

Κώδικας 5.6: Δείγμα κώδικα απο το labeled-char-rnn

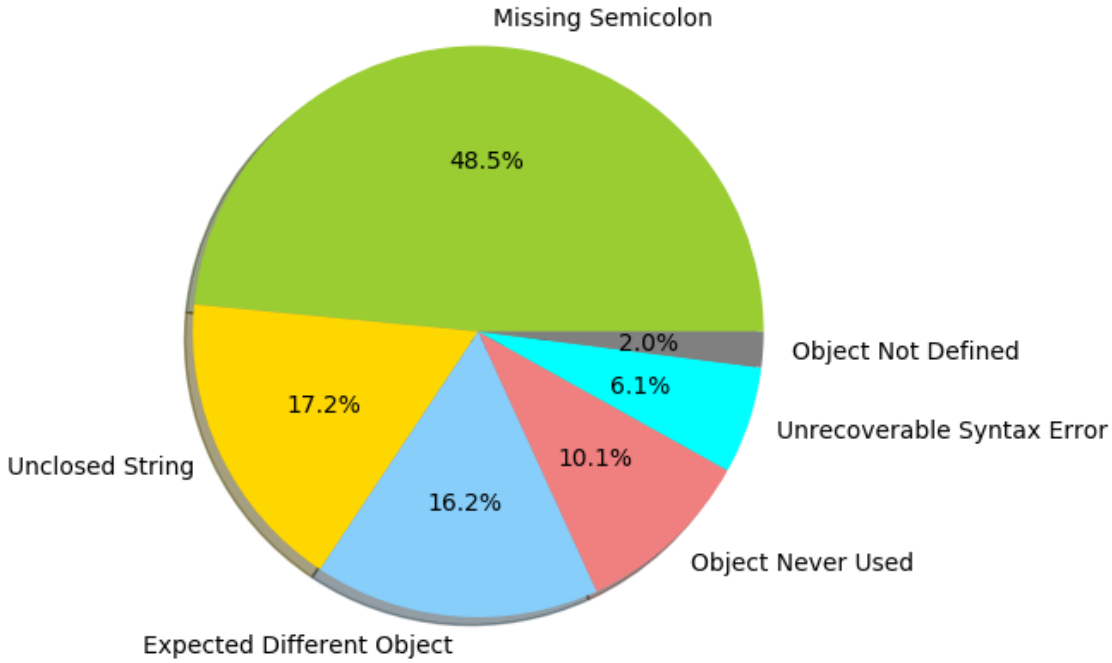
Στα σχήματα 5.9 και 5.10 μπορούμε να συγκρίνουμε τις επιδόσεις των δύο μοντέλων. Το dataset αυτό περιέχει κατά μέσο όρο μικρότερα αρχεία οπότε και τα μεγέθη των παραγώμενων αρχείων είναι μικρότερα (113 γραμμές char-rnn – 47 γραμμές labeled-char-rnn). Και πάλι, το labeled-char-rnn μοντέλο κλείνει με μεγαλύτερη συνέπεια τα αρχεία του. Τα ποσοστά ολοκλήρωσης της συντακτικής ανάλυσης και οι μέσοι όροι λαθών δείχνουν βελτίωση σε σχέση με το προηγούμενο σετ δεδομένων, ενώ τα παρόντα μοντέλα χρησιμοποιούν σημαντικά λιγότερες παραμέτρους. Ωστόσο, και σε αυτό το πείραμα το labeled-char-rnn μοντέλο αδυνατεί να χρησιμοποιήσει την παραπάνω πληροφορία που του δίνεται. Τα μοτίβα των συνηθισμένων λαθών επαναλαμβάνονται (5.11 - 5.14)



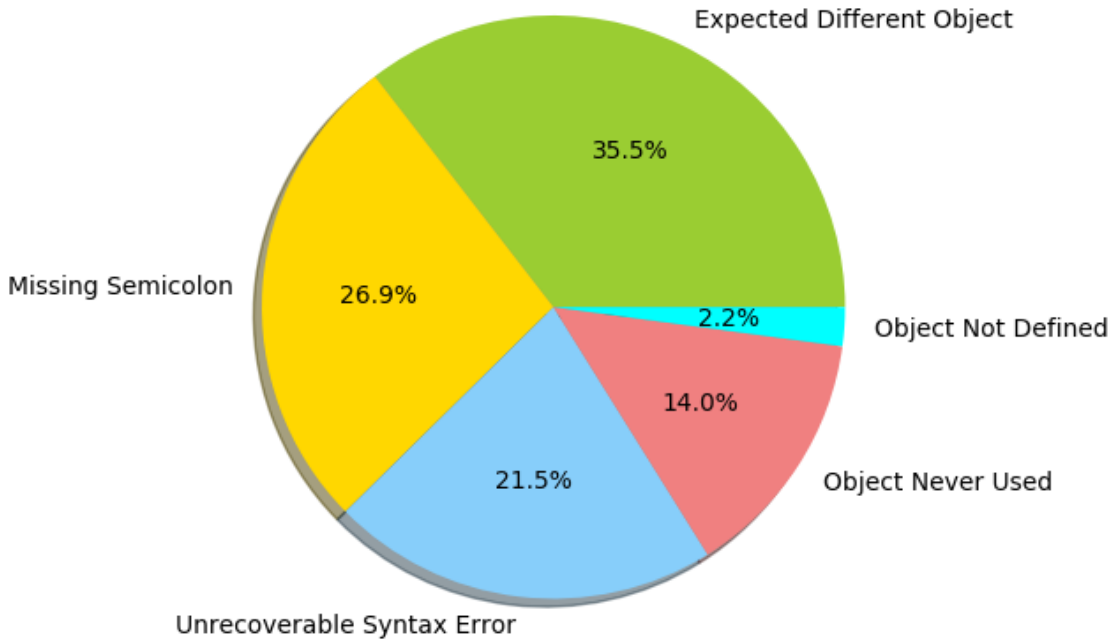
Σχήμα 5.9: Στατική ανάλυση κώδικα για τα αποτελέσματα του char-rnn μοντέλου



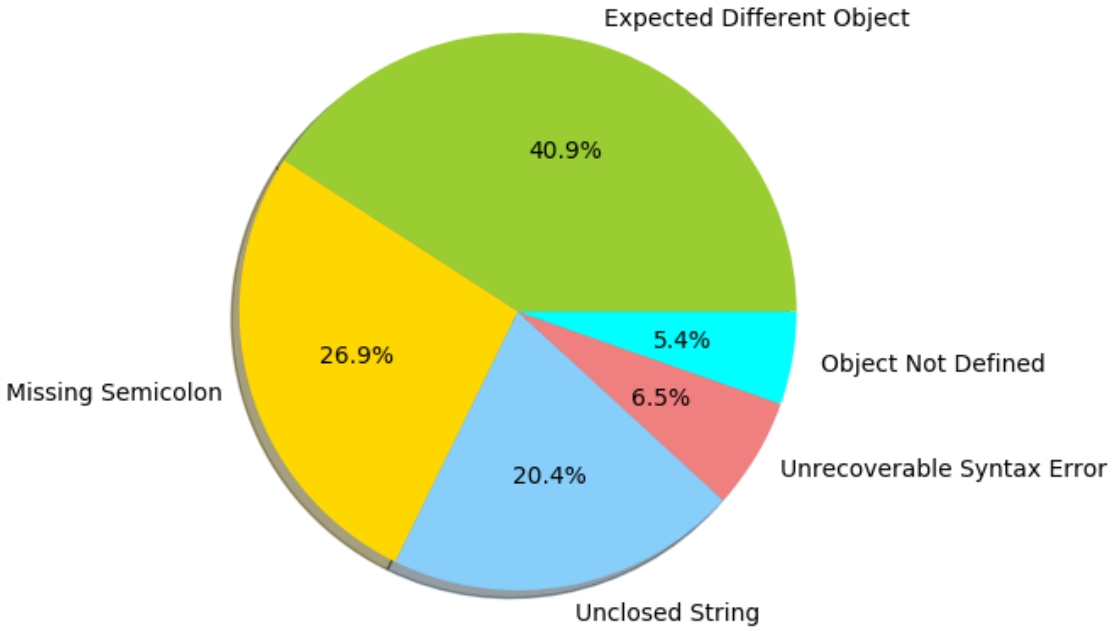
Σχήμα 5.10: Στατική ανάλυση κώδικα για τα αποτελέσματα του labeled-char-rnn μοντέλου



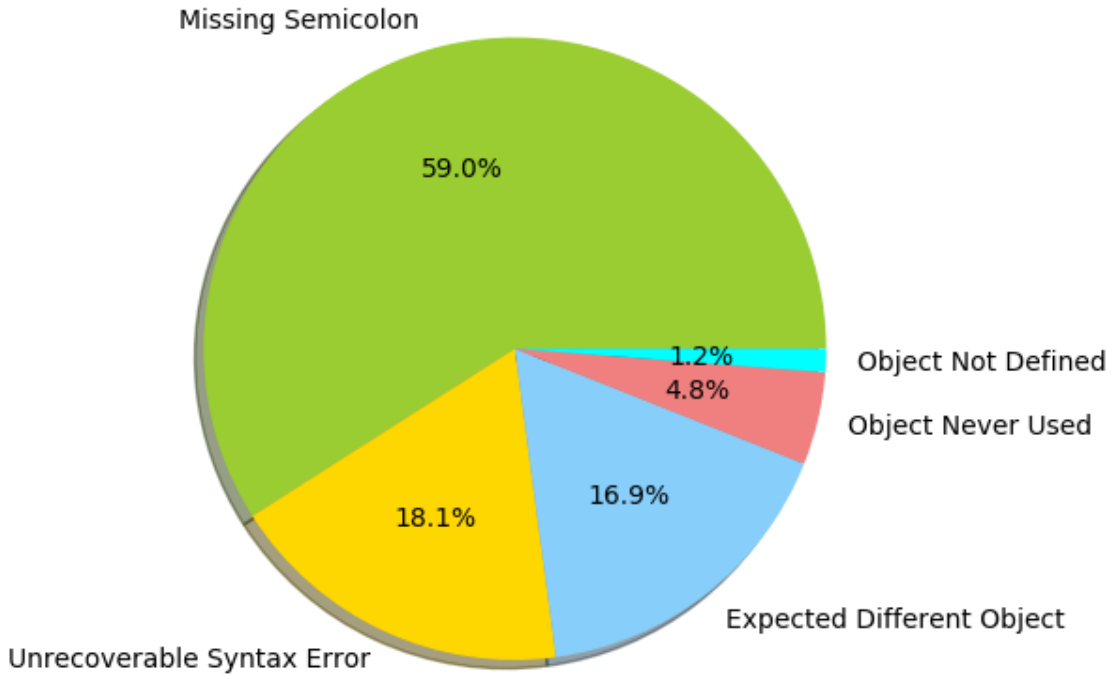
Σχήμα 5.11: Συνηθέστερο λάθος των αρχείων του char-rnn



Σχήμα 5.12: Δεύτερο συνηθέστερο λάθος των αρχείων του char-rnn



Σχήμα 5.13: Συνηθέστερο λάθος των αρχείων του labeled-char-rnn



Σχήμα 5.14: Δεύτερο συνηθέστερο λάθος των αρχείων του labeled-char-rnn

5.2.3 100 δημοφιλέστερα Github Javascript Projects Παραγώμενος Κώδικας με ρυθμισμένη δειγματοληψία

Η μέθοδος παραγωγής κώδικα βασίζεται στη δειγματοληψία της εξόδου του συστήματος μας. Με σκοπό τη διερεύνηση της λειτουργίας των δύο μοντέλων, θα ρυθμίσουμε την κατανομή πιθανότητας που προτείνει το μοντέλο, ώστε να είναι πιο σίγουρο για τις προβλέψεις του. Χρησιμοποιούμε την συνάρτηση Softmax Temperature και επιλεγμένη τιμή για τη θερμοκρασία 0.85. Επιλέγουμε τα μοντέλα που εκπαιδεύτηκαν στο Github js dataset και τον ίδιο κώδικα αρχικοποίησης που χρησιμοποιήσαμε παραπάνω. Οι κώδικες 5.7, 5.8 περιέχουν αρχεία που παρήχθησαν από τα 2 μοντέλα. Στο κόστος της ποικιλίας του παραγώμενου κώδικα, τα μοντέλα κάνουν πιο "συνηθισμένες" επιλογές.

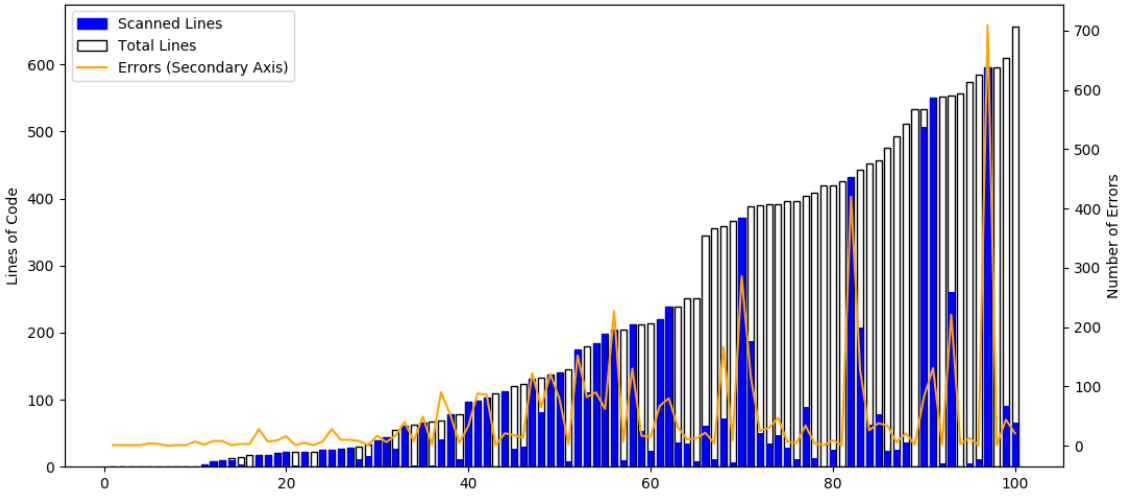
```
WebInspector.ExtensionServerView.scanUserAgentStyles = function (
    vertexStyle, allStyleSheets) {
    var verticalStylesIdentifier = (styleSheetIds.length);
    if (!videoStyleSheetIdsMap)
        return;
    var apiSetting = new WebInspector.StylesSidebarPanes();
    if (!rule)
        return;
    var rules = allSubStylesRules.slice();
    if (cssModel)
        cssView = rules;
    var textStyleSheet = rule.styleSheet.styleSheet;
    if (!encoding)
        return;
    this.tasks().sanitizeText(styleSheet);
    return this._rulesSchema.callTargetAdded({
        textNode: this._parentNode.classList.contains("text"),
        value: StyleSheetName,
        headers: checkedStyles
    });
}, __proto__: WebInspector.VBox.prototype
}
```

Κώδικας 5.7: Δείγμα κώδικα από το char-rnn με ρυθμισμένη δειγματοληψία

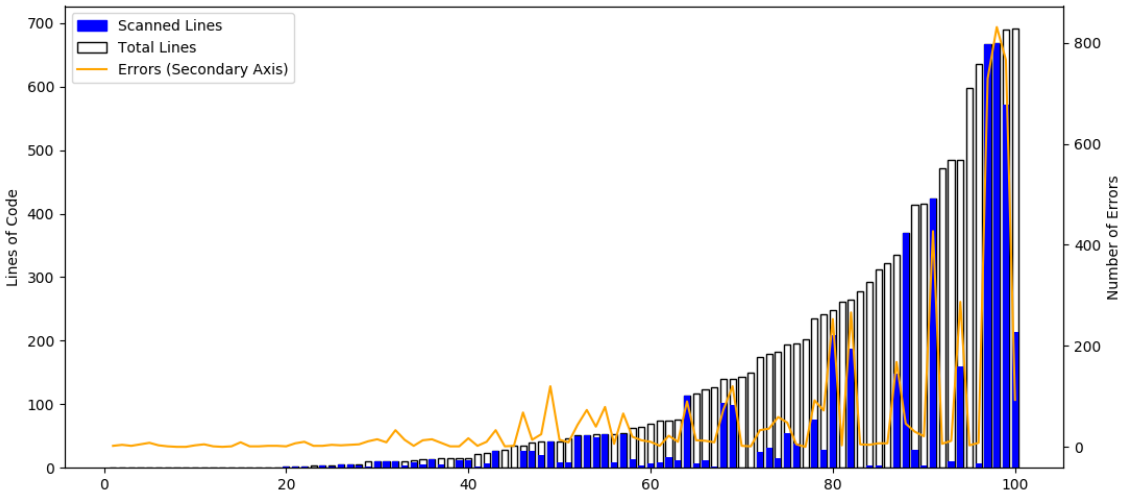
```
addProvider(tty, Query, pdf, readJson)
}
export class flushedConstants(opts, cb) {
    readJson(targets, options[tar], nonLock, false, long)
}

function xhr() {
    try {
        readJson(path.join(__dirname, '_location'), files)
    }
    console.log(console.log('Error ' + pjType == 'Progress"),'
            AllTimes below 0 / all commands are sure that do not exist.
            '
            ''
            reload your app with initialization calls and the `npm install`
            should fail '
            prefunction(resolve, reject) {
                fs.existsSync(resolveReadFileExists)
            }
            protocetins getFerrors() {
                var resolved = workers =
                    return log.verbose('resolved', '--execute')
                validate('Assets', files.map(function (file) {
                        if (!file.indexOf('.')) {
                            result = result.map(function (file) {
                                result.path = file
                                ignored + expressed(path.join(__dirname, '
                                    ..', '.'))
                            }).map(function (file) {
                                if (function (name) {
                                        return file.type === 'POS' || file.
                                            path.replace(/\}/g, '@')
                                    })
                            })
                        })
                })
            })
    }
```

Κώδικας 5.8: Δείγμα κώδικα απο το labeled-char-rnn με ρυθμισμένη δειγματοληψία

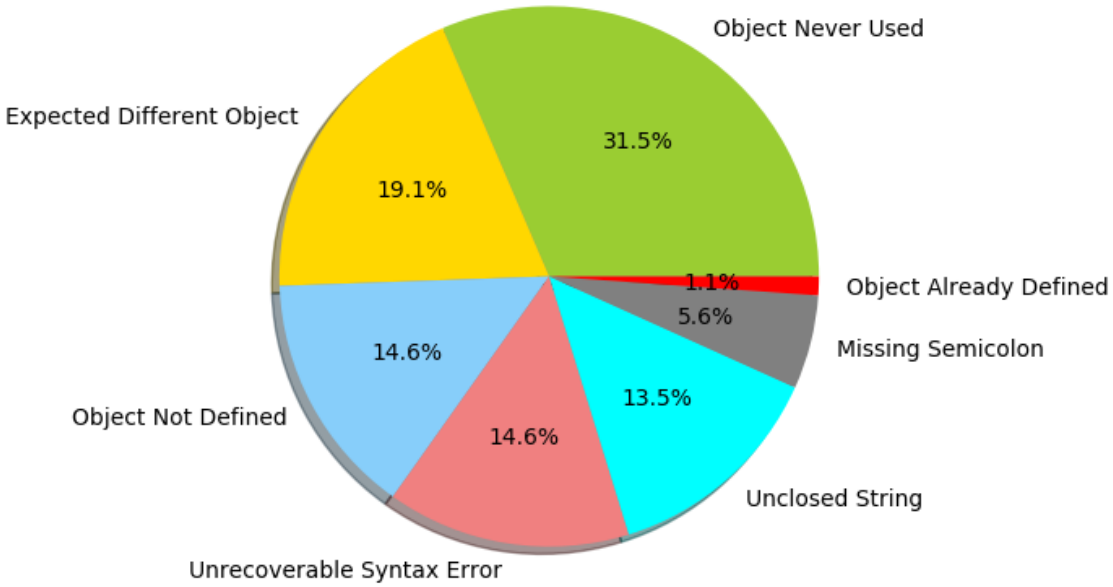
Στις εικόνες 5.15 - 5.20 παρατίθενται τα αποτελέσματα της ανάλυσης. Το μέσο μήκος των παραγώμενων αρχείων ανεβαίνει σημαντικά σε σχέση με τις προηγούμενες προσεγγίσεις (218 γραμμές char-rnn – 132 γραμμές labeled-char-rnn). Όπως και στα προηγούμενα αποτελέσματα, το μοντέλο labeled-char-rnn κλείνει με μεγαλύτερη συνέπεια τα αρχεία του, αλλά τα παραγώμενα αρχεία ολοκληρώνουν σε μικρότερο ποσοστό τον συντακτικό έλεγχο επιτυχημένα. Ενδιαφέρουσα είναι η εμφάνιση του λάθους Object Already Defined στα πιο συνηθισμένα λάθη. Πράγματι, τα αρχεία που παράγονται από τα μοντέλα, τα οποία έχουν μεγαλύτερη πεποίθηση για τις προβλέψεις τους, τείνουν να επαναλαμβάνουν διάφορες δομές.



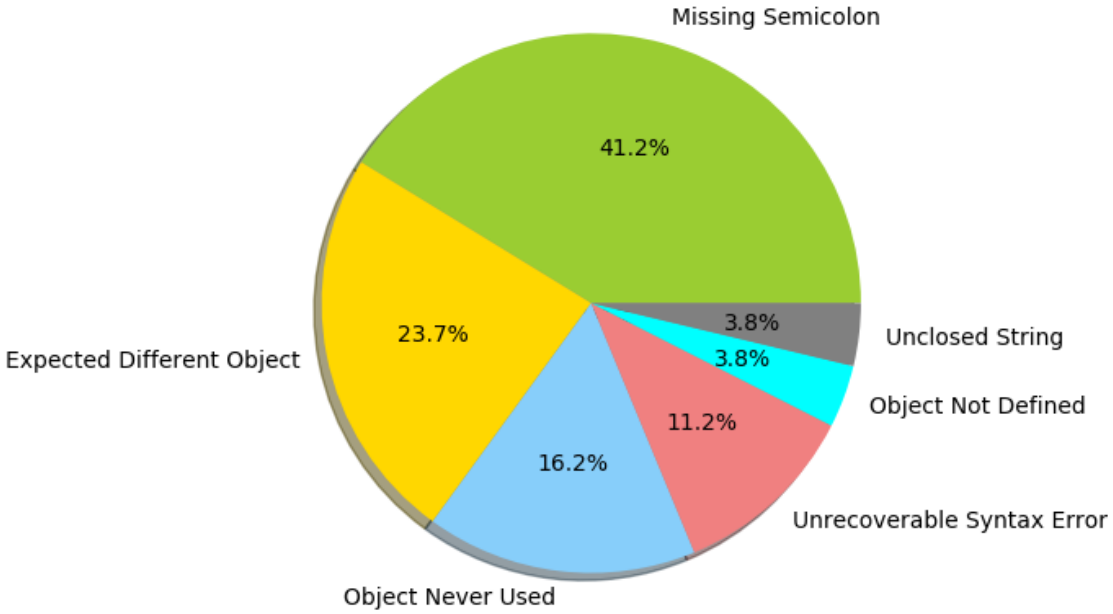
Σχήμα 5.15: Στατική ανάλυση κώδικα για τα αποτελέσματα του char-rnn μοντέλου



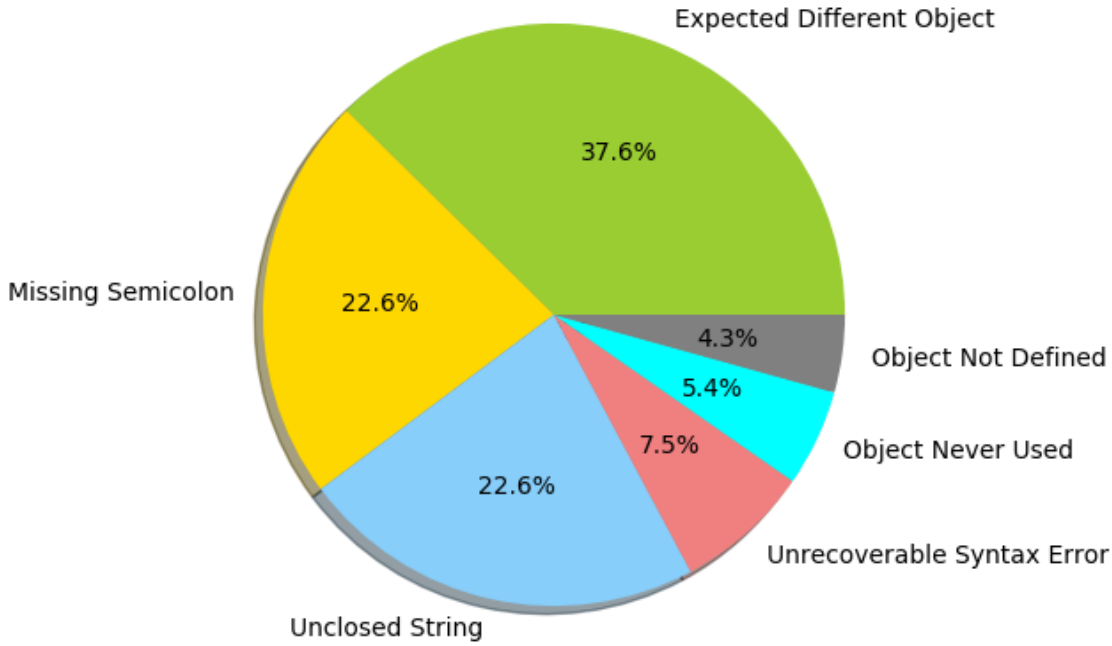
Σχήμα 5.16: Στατική ανάλυση κώδικα για τα αποτελέσματα του labeled-char-rnn μοντέλου



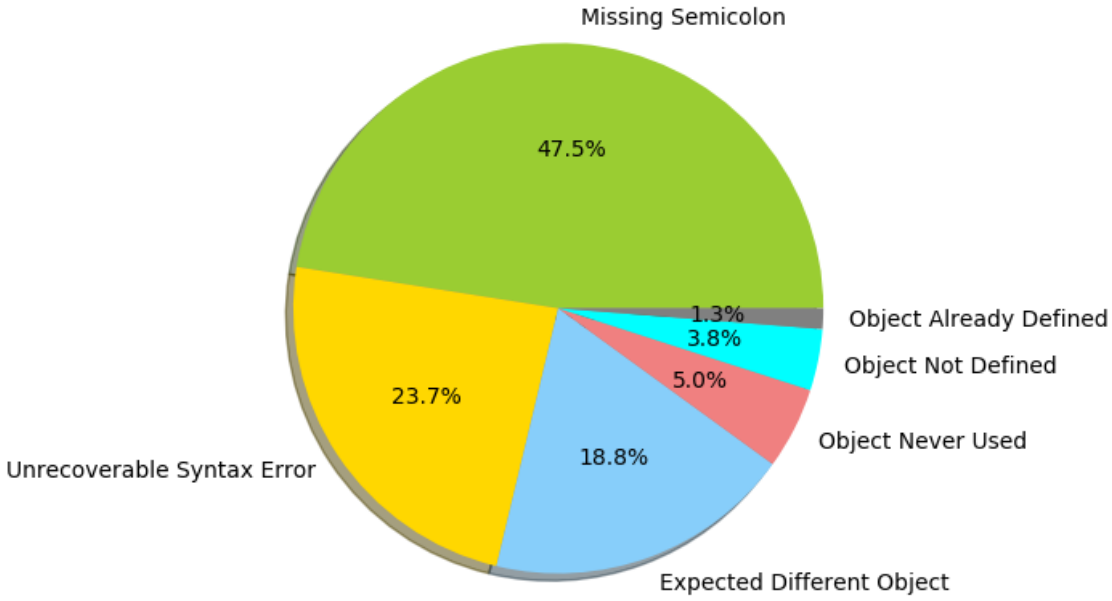
Σχήμα 5.17: Συνηθέστερο λάθος των αρχείων του char-rnn



Σχήμα 5.18: Δεύτερο συνηθέστερο λάθος των αρχείων του char-rnn



Σχήμα 5.19: Συνηθέστερο λάθος των αρχείων του labeled-char-rnn



Σχήμα 5.20: Δεύτερο συνηθέστερο λάθος των αρχείων του labeled-char-rnn

5.2.4 Σχόλια περί σύγκρισης

Τα αποτελέσματα της παραγωγής κώδικα που δεν είναι πραγματικά λειτουργικός είναι δύσκολο να ποσοτικοποιηθούν. Η εγγενής δυσκολία ερμήνευσης της εκμάθησης των νευρωνικών δικτύων, δυσκολεύει περαιτέρω την προσπάθεια αυτή. Με σκοπό την αποσαφήνιση των αποτελεσμάτων, δημιουργούμε μία απλή μετρική ώστε να συγκρίνουμε τα 6 μοντέλα που δοκιμάστηκαν.

$$M = Errors_{avg}/(Lines_{avg} * \%Scanned_{avg}) \tag{5.1}$$

Πίνακας 5.5: Τιμές της Μετρικής M (Μικρότερες τιμές του M είναι προτιμότερες)

	char-rnn	labeled-char-rnn
Github	0.61	0.86
NPM	0.64	0.77
Github Temperature	0.38	0.52

Στη σχέση 5.1 $Errors_{avg}$ είναι ο μέσος όρος λαθών των αρχείων, $Lines_{avg}$ είναι ο μέσος όρος γραμμών των αρχείων και $\%Scanned_{avg}$ ο μέσος όρος του ποσοστού ολοκλήρωσης συντακτικής ανάλυσης των αρχείων Από τον πίνακα 5.5 μπορούμε να εξάγουμε 2 σημαντικές παρατηρήσεις. Η πρώτη είναι πως το μοντέλο labeled-char-rnn έχει σταθερά χειρότερες επιδόσεις στην παραγωγή ποιοτικού κώδικα, παρόλο που έχει στη διάθεση του παραπάνω πληροφορία για τον κώδικα και μάλιστα σημαντική. Αυτό οφείλεται εν μέρει στην ταυτόχρονη στοχαστική επιλογή χαρακτήρα και είδους του, που μπορεί να επινοεί συνδυασμούς που δεν έχει ξαναδεί. Κατά δεύτερο λόγο, η στοχαστικότητα που εισάγεται για την παραγωγή κώδικα δυσκολεύει γενικότερα τα μοντέλα να γράψουν ορθά προγράμματα, για αυτό και όταν επιβάλλουμε στα συστήματα να εμπιστεύονται τις προβλέψεις τους, παίρνουμε καλύτερες τιμές για τη μετρική M.

Κεφάλαιο 6

Συμπεράσματα και Μελλοντική Εργασία

6.1 Συμπεράσματα

Ας ξεκινήσουμε την εξαγωγή συμπερασμάτων της διπλωματικής εργασίας, ανατρέχοντας στους στόχους που τέθηκαν στο πρώτο κεφάλαιο, αλλά με αντίθετη φορά. Ιδανικά, θα θέλαμε τα μοντέλα που δημιουργήσαμε να παράγουν κώδικα που "κάνει" κάτι χρήσιμο. Αν και αυτό δεν αποκλείεται να συμβεί – χάρη στη δυνατότητα της προσέγγισης να παράγει κώδικα επ΄ άπειρον – δεν μπορούμε να το εγγυηθούμε, ούτε να το ελέγξουμε. Εντοπίζουμε λοιπόν την πρώτη αδυναμία της προσέγγισης στην έλλειψη ικανότητας άμεσης οδήγησης των συστημάτων και στη στοχαστική τους φύση.

Έπειτα ενδιαφερόμαστε να εξετάσουμε κατά πόσο ο κώδικας αυτός μπορεί να μεταφραστεί και να μην έχει συντακτικά λάθη. Αν και μεμονωμένα κομμάτια κώδικα πληρούν αυτές τις προϋποθέσεις, το σύστημα υποπίπτει συχνά σε τέτοια λάθη και κανένα αρχείο δεν παράχθηκε που να είναι συντακτικά και προγραμματιστικά ορθό. Ακόμα και όταν τα νευρωνικά δίκτυα ενημερώνονται απλοϊκά (labeled models) με πρότερη γνώση για τον κώδικα και τη φύση του, δεν εγγυόμαστε πως αποτυπώνουν εύρωστα συμπεράσματα. Είναι τέτοια η φύση της κατά χαρακτήρα, end-to-end RNN προσέγγισης που στα πλαίσια του αυτόματου προγραμματισμού αδυνατεί να πετύχει το, γνωστό στη βιβλιογραφία ως, Ground Learning (Βασική Μάθηση).

Ο απλούστερος στόχος που τέθηκε ήταν το προϊόν των συστημάτων μας να μοιάζει με κώδικα. Αν και στόχος που δεν είναι εύκολο να ποσοτικοποιηθεί, είναι εμφανές τόσο από τα δείγματα όσο και από το σύνολο των παραγώμενων αρχείων πως όλα τα συστήματα που δοκιμάσαμε το καταφέρνουν. Σε αντίθεση με το ground learning η δυνατότητα των συστημάτων για μίμηση είναι εντυπωσιακή. Ονόματα μεταβλητών που υπακούν στους κανόνες της αγγλικής και τις προγραμματιστικές συμβάσεις, χρήση λογικών δομών και ενδιαφέρουσες αλφαριθμητικές ακολουθίες μαρτυρούν τη αναπαραστατική δύναμη των "βαθιών" αναδραστικών νευρωνικών δικτύων.

Σε ότι αφορά την εξέταση των labeled μοντέλων και της διαίσθησης ότι περισσότερη πληροφορία θα σημαίνει καλύτερα αποτελέσματα κώδικα, η πραγματικότητα είναι διαφορετική. Η

προσέγγιση μας, δεν καταφέρνει να αυξήσει την ποιότητα του παραγώμενου κώδικα, αντίθετα συχνά βοηθάει προς την αντίστροφη κατεύθυνση. Σημειώνεται πως η ταυτόχρονη δειγματοληψία χαρακτήρα και είδους χαρακτήρα φαίνεται να δυσκολεύει την παραγωγή κώδικα. Αυτό σημαίνει πως η πρότερη γνώση σύνταξης και σημασιολογίας θα πρέπει να δοθεί στο σύστημα με πιο αποτελεσματικό τρόπο.

Ο αυτόματος προγραμματισμός είναι ένας εξαιρετικά δύσκολος στόχος. Αναμέναμε, λοιπόν, πως οι απλές αυτές προσεγγίσεις με αναδραστικά νευρωνικά δίκτυα δε θα λύσουν το πρόβλημα. Η προσπάθεια παραγωγής κώδικα όμως, είναι ένας ενδιαφέρον τρόπος να εξετάσουμε την ισχύ τους. Χάρη στην πλούσια δυναμική, την δυνατότητα για εκμάθηση σύνθετων αναπαραστάσεων, την αγνωστικότητα ως προς την είσοδο και την έξοδο αλλά και τη δυνατότητα διαχείρισης ακολουθιών είναι ένα σημαντικό εργαλείο στο οπλοστάσιο της Μηχανικής Μάθησης.

6.2 Μελλοντική Εργασία

Το σύστημα που αναπτύχθηκε στα πλαίσια αυτής της διπλωματικής θα μπορούσε να εξελιχθεί περαιτέρω, τουλάχιστον ως προς τρεις κατευθύνσεις.

Αρχικά έχει ενδιαφέρον η προσπάθεια παραγωγής κώδικα και με διαφορετικές δομές μηχανικής εκμάθησης. Πιο συγκεκριμένα, πρόσφατα αναπτύχθηκαν τα generative adversarial networks [4] (ανταγωνιστικά δίκτυα παραγωγής), δομές στις οποίες το ένα σύστημα νευρωνικών δικτύων παράγει και το δεύτερο κρίνει αυτό που παρήγαγε το πρώτο. Αυτή η ιδέα πρέπει να δοκιμαστεί στα πλαίσια του αυτόματου προγραμματισμού.

Έπειτα, όπως αναφέραμε, η δομή αναδραστικών δικτύων που χρησιμοποιήθηκε στα πλαίσια της διπλωματικής εργασίας αυτής είναι μία απλοϊκή προσέγγιση. Το ίδιο ισχύει και για τρόπο με τον οποίο δίνουμε στο σύστημα πρότερη γνώση για τη γλώσσα και τη δομή της. Δεδομένου του ανεξερεύνητου χώρου έρευνας γύρω από τα νευρωνικά δίκτυα, μπορούμε να δοκιμάσουμε πιο σύνθετες δομές εκμάθησης και να δίνουμε με διαφορετικό τρόπο την πρόσθετη πληροφορία για τον κώδικα. Μια πρώτη τέτοια ιδέα είναι η χρήση των tree-LSTM δικτύων [20] για την διαχείριση των abstract syntax trees της γλώσσας που χρησιμοποιείται.

Τέλος, η μοντελοποίηση και η πρόβλεψη κατά χαρακτήρα, στα πλαίσια του προγραμματισμού, παρουσιάζουν χρήσιμες ιδιότητες, τις οποίες θα μπορούσαμε να χρησιμοποιήσουμε για να αναπτύξουμε βοηθήματα ανάπτυξης λογισμικού. Συστήματα στατικής ανάλυσης και διόρθωσης κώδικα αλλά και βοηθήματα συμπλήρωσης κώδικα μπορούν να δημιουργηθούν με βάση τις προσεγγίσεις που περιγράφηκαν παραπάνω.

Βιβλιογραφία

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *ICLR*, 2014.

[2] J. Bergstra and Y. Bengio. Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 2012.

[3] A. W. Biermann. Automatic programming : A tutorial on formal methodologies. *Journal of Symbolic Computation*, 1985.

[4] I. Goodfellow, J. Pouget-Abadie, M. Mirza, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. *NIPS*, 2014.

[5] A. Graves. Generating Sequences with Recurrent Neural Networks. *Technical Reports*, 2013.

[6] S. Hochreiter and J. J. Schmidhuber. Long short-term memory. *Neural Computation*, 1997.

[7] A. Joulin and T. Mikolov. Inferring Algorithmic Patterns with Stack-Augmented Recurrent Nets. *arXiv*, 2015.

[8] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.

[9] P. Liu, X. Qiu, and X. Huang. Recurrent Neural Network for Text Classification. *arXiv*, 2016.

[10] V. Murali, S. Chaudhuri, and C. Jermaine. Bayesian Sketch Learning for Program Synthesis. *arXiv*, 2017.

[11] E. Parisotto, A.-R. Mohamed, R. Singh, L. Li, D. Zhou, and P. Kohli. Neuro-Symbolic Program Synthesis. *arXiv*, 2017.

[12] D. L. Parnas. Software aspects of strategic defense systems. *ACM SIGSOFT Software Engineering Notes*, 1985.

[13] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training Recurrent Neural Networks. *arXiv*, 2012.

[14] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 2006.

[15] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 2014.

[16] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus. End-To-End Memory Networks. *NIPS*, 2015.

[17] I. Sutskever. Training Recurrent neural Networks. *PhD thesis*, 2013.

[18] I. Sutskever, J. Martens, and G. Hinton. Generating Text with Recurrent Neural Networks. *ICML*, 2011.

[19] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *NIPS*, 2014.

[20] K. S. Tai, R. Socher, and C. D. Manning. Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks. *ACL*, 2015.

[21] P. Yin and G. Neubig. A Syntactic Neural Model for General-Purpose Code Generation. *arXiv*, 2017.